Towards a robust, effective and resource efficient machine learning technique for IoT security monitoring.

ZAKARIYYA, I.

2022

Towards a Robust, Effective and Resource Efficient Machine Learning Technique for IoT Security Monitoring

Idris Zakariyya

A thesis submitted in partial fulfilment
of the requirements of
Robert Gordon University
for the degree of Doctor of Philosophy

December 2022

Abstract

Internet of Things (IoT) devices are becoming increasingly popular and an integral part of our everyday lives, making them a lucrative target for attackers. These devices require suitable security mechanisms that enable robust and effective detection of attacks. Machine learning (ML) and its subdivision Deep Learning (DL) methods offer a promise, but they can be computationally expensive in providing better detection for resource-constraint IoT devices. Therefore, this research proposes an optimization method to train ML and DL methods for effective and efficient security monitoring of IoT devices. It first investigates the feasibility of the Light Gradient Boosting Machine (LGBM) for attack detection in IoT environments proposing an optimization procedure to obtain its effective counterparts. The trained LGBM can successfully discern attacks and regular traffic in various IoT benchmark datasets used in this research. As LGBM is a traditional ML technique, it may be difficult to learn complex network traffic patterns presents in IoT datasets. Therefore, we further examine Deep Neural Networks (DNNs), proposing an effective and efficient DNN-based security solution for IoT security monitoring to leverage more resource savings and accurate attack detection. Investigation results are promising as the proposed optimization method exploits the mini-batch gradient descent with simulated micro-batching in building effective and efficient DNN-based IoT security solutions. Following the success of DNN for effective and efficient attack detection, we further exploit it in the context of adversarial attack resistance. The resulting DNN is more resistant to adversarial samples better than its benchmark counterparts and other conventional ML methods. To evaluate the effectiveness of our proposal, we considered on-device learning in federated learning settings using decentralized edge devices to augment data privacy in resource-constrained environments. To this end, the performance of the method was evaluated against various realistic IoT datasets (e.g., NBaIoT, MNIST) on virtual and realistic testbed set-up with GB-BXBT-2807 edge-computing-like devices. The experimental results show that the proposed method can reduce memory and time usage by 81% and 22% in the simulated environment of virtual workers compared to its benchmark counterpart. In the realistic testbed scenario, it saves 6% of memory footprints with a reduction of execution time by 15% while maintaining a better and state-of-the-art accuracy.

Keywords: Internet of Things Security, Machine Learning, Deep Learning, Resource-constraint, Attack Detection, Federated Learning, Testbed.

Declaration of Authorship

I declare that I am the sole author of this thesis and that all verbatim extracts contained in the thesis have been identified as such and all sources of information have been specifically acknowledged in the bibliography. Parts of the work presented in this thesis have appeared in the following publications.

1. Idris Zakariyya; Al-Kadri, M. Omar; Kalutarage, Harsha; Petrovski, Andrei, Reducing Computational Cost in IoT Cyber Security: Case Study of Artificial Immune System Algorithm. SECRYPT, 2019: 523-528. https://dblp.org/rec/conf/icete/ZakariyyaAKP19 (chapter 3)

2. Idris Zakariyya; Al-Kadri, M. Omar; Kalutarage, Harsha, Resource Efficient Boosting Method for IoT Security Monitoring, 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), 2021, pp. 1-6, doi: 10.1109/C-CNC49032.2021.9369620. (chapter 3)

3. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Robust, Effective and Resource Efficient Deep Neural Network for Intrusion Detection in IoT Networks, in CPSS'22, in 17th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2022), https://doi.org/10.1145/3494107.3522772 (chapter 4 and 5)

4. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Memory Efficient Federated Deep Learning for Intrusion Detection in IoT Networks, in AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, http://ceur-ws.org/Vol-3125/paper7.pdf (chapter 6)

5. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Resource Efficient Federated Deep Learning for IoT Security Monitoring, Book Chapter in ADIoT, in 27th European Symposium on Research in Computer Security (ESORICS 2022), pp 122–142, Lecture Notes in Computer Science, vol 13745. Springer, Cham. https://link.springer.com/chapter/10.1007/978-3-031-21311-3_6. (chapter 6)

6. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Towards a Robust, Effective and Resource Efficient Deep Learning Technique for IoT Security Monitoring, Computers & Security, Under Review.

Acknowledgements

I want to use this opportunity to offer my warmest gratitude and appreciation to my supervisory team Dr Harsha Kalutarage and Dr Omar Al-kadri. Both of you have supported me academically with the best advice instantly. In particular, Dr Harsha Kalutarage motivates me and inspires me to be a better researcher, particularly in the aspect of Artificial Intelligence (AI) and Internet of Things (IoT) cybersecurity monitoring. Your support, motivational advice and inspirations are always commendable.

I also thank the Petroleum Technology Development Fund (PTDF) for funding this research work. Your financial support and commitment throughout my study are commendable.

Finally, I must thank my parent, particularly my mother for the support and care she gives me to achieve the best in my life. Your legacy of making me who I am today shall always be remembered.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The Internet of Things (IoT) consists of Internet-enabled devices that use embedded systems such as processors, sensors and communication hardware to collect and exchange data. The data collected by IoT devices worldwide can reach 73.1 zettabytes by 2025 [1]. With its potential to revolutionize the interaction between objects using the recent advances in super-cheap computer chips and the ubiquity of wireless networks, it is possible to make anything as small as a contact lens or as big as an aeroplane to be part of the IoT. As IoT represent an ecosystem of devices used in smart homes, smart cities, and many intelligent automation systems, IHS Markit estimates that 125 billion devices will be connected to the IoT by 2030 [2]. Most of these devices combine Artificial Intelligence (AI) with IoT infrastructure to enable more efficient IoT operations, improve human-machine interaction, and enhance data management and analytic. Therefore they can be considered as Artificial Intelligence of Things (AIoT) [3]. The gradual provision of such devices is transforming the world into a sophisticated interconnected domain. This is good as it offers modern user convenience and a source for generating profits by prominent industries such as Google and Alexa [4]. But on the other hand, it can open up a broader attack surface. Attackers can exploit vulnerabilities in software/hardware or embedded AI of these devices, especially when they are connected to the external world to launch a cyber disaster. In October 2016, for example, attackers launched a wave of IoT botnet attacks called Mirai. They used various IoT devices to deny access to high-profile websites like Twitter, Amazon, Github, and Netflix [5]. A study by Venafi has revealed that 76% of IoT devices are vulnerable to cyber attacks, while only 24% of these devices encrypt their data before transmission [6]. On the other hand, the hardware of these constrained devices can accommodate 32KB - 128KB units of Random Access Memory (RAM) with 256KB - 512KB of embedded flash memory [7]. For these reasons, security challenges in IoT must be addressed with robust, effective and resource-efficient detection techniques.

Recent research has shown the capabilities of AI technologies, particularly Machine Learning (ML) based solutions for cyber security monitoring [8]. This is due to ML's capacity of developing a model that can learn the statistical distribution of various datasets to make predictions without the need to explicitly write a set of rules. However, the procedure of building such ML methods is computationally expensive with complex datasets [9], with intensive memory and time resources requirements [10]. Because of that, ML techniques require attentive optimization to scale through an IoT-like resource-constrained environment. Therefore, this thesis starts with the investigation of using feature reduction techniques to optimize ML algorithms. The assumption is that feature reduction can reduce the cost of training ML algorithms with a given dataset. Then, it proposed an optimization procedure that can produce a lightweight ML technique which utilizes relatively minimal memory and execution time while accurately discerning attacks and regular traffic on IoT networks. The evaluation procedure used a Light Gradient Boosting Method (LGBM) baseline model from the Decision Tree Ensemble Method (DTEM) conventional ML with our optimization procedure to obtain its resource-efficient counterparts.

The success in optimizing the LGBM technique to be more resource efficient motivates further investigation of other AI technologies, especially the Deep Neural Network (DNN) based methods. Recent research has shown the capabilities of DNN in intrusion detection, which can outperform most of the ML models in cyber security monitoring [11]. A disadvantage of DNN-based methods, however, is that they require a lot of resources to build a model that can provide better detection with a multi-dimensional feature set [12]. This would be problematic in training scenarios such as edge machine learning, in which smart devices can process data locally using machine and deep learning algorithms (e.g. federated learning). Moreover, compared to mainstream IT devices, IoT devices are equipped with limited computing resources (processing and storage) to enable maximum data output with minimum energy requirements while remaining cost-effective. As a result, DNN-based security solutions designed for mainstream IT devices cannot simply be deployed for security monitoring in an environment with limited computing resources.

In addition, the detection capabilities of DNN-based methods can easily be exploited by feeding the network with adversarial samples [13]. Considering these strengths and limitations of ML and DNN methods in IoT security monitoring, this research aims to address these challenges by exploiting AI techniques to consider their optimization to reduce computational complexity, memory and time requirements at training and testing and their resilience to adversarial attacks. This can be an appropriate security solution in an environment with limited computational resources that requires efficient detection of attacks. In that context, the thesis investigates the following research questions (RQs) to develop an appropriate DNN-based method for the security monitoring of resource-constrained IoT devices.

RQ1: What are the existing methods that can be used to train and build efficient and effective ML and DNN methods in a resource-constrained environment? (see section 2.3)

RQ2: How to train existing ML and DNN methods to be resource efficient in security monitoring of resource constrained environments like IoT? (see chapters 3, 4 6)

RQ3: How to train the DNN algorithms in RQ2 to be robust against adversarial machine learning attacks while maintaining their resource constrained nature? (see chapter 5)

1.2 Research Objectives and Approaches

Following earlier research conducted in the field of IoT security, there is a demand for effective, efficient and robust AI security methods in a resource-constrained environment. This is due to IoT resource limitations (memory and processor) and existing AI-based cyber security methods that can be resource-hungry to process complex multidimensional data. As such, findings from this research can help security practitioners and industries about deploying secure, robust and efficient AI solutions in a resource-constrained environment. In addition, other cyber security researchers can utilize the methods proposed in this thesis to improve existing AI security solutions in IoT network environments. Therefore, this work identifies five key objectives as follows:

RO1: An in-depth literature survey of IoT security monitoring techniques and their limitations. (RQ1)

RO2: Develop an optimization algorithm that can utilize less computational parameters of DTEM to build a lightweight detection model for IoT security monitoring. (RQ2)

RO3: Develop an optimization algorithm while utilizing weight elimination, data parallelism, micro-batching and regularization techniques, to train DNN in a robust, effective and resource-efficient manner. (RQ2 and RQ3)

RO4: Evaluate the DNN-based method performance in a realistic environment using multiple decentralized IoT like devices. (RQ2)

For investigating the effectiveness of ML and DNN methods, this research adopts a deductive approach that involves experiments with realistic publicly available IoT benchmark datasets for hypothesis validation. Reduction of computational resource consumption of running ML algorithms for IoT security has contributed to other contributions of this research, for example on-device learning. This served as proof of concepts to address the carefully identified research questions listed in section 1.1. Experimental evaluations are designed to build generic and appropriate AI-based security solutions for IoT. Therefore, we evaluated the possibility of using feature reduction techniques to reduce the computational complexity (memory and time) of running ML, in particular the Artificial Immune System (AIS) for IoT network traffic classification. AIS was selected as a traditional ML algorithm for our experiments because it can be resource-hungry in processing multidimensional data [14]. For the proof of concept, the Gini Index (GI) and Principal Component Analysis (PCA) are adopted and tested with the AIS resource-hungry algorithm, and their resource-minimization capability in the IoT network environment is compared. This served as an initial step toward answering RQ2.

In addition, to address RQ2 based on RO2, a further step to introduce a novel optimization algorithm that can train ML models to save computational resources (execution time and memory) in training and testing with better detection is considered. The rationale is to explore the trade-off between balancing the accuracy of ML detection and resource reduction. Because the algorithm evaluation considered all data features, it can be used as a pipeline for optimizing ML schemes without reducing the features of their training data. As such, a lightweight ML method using the LGBM approach can be proposed. The advantage of this proposal is that LGBM is a promising ML model that can perform better for classification tasks [15]. Therefore, reducing its computational cost can produce a more lightweight counterpart with minimal resource consumption that may scale across IoT devices. As such, RO2 targets improving the efficiency and effectiveness of traditional ML models to propose adaptable IoT security solutions.

The success of AI-based (DNN) security solutions in terms of accurate classification using multi-sensory data seems promising [16]. This is due to their feature learning capability that allows them to learn the raw traffic features captured from various cyber security datasets. DNN can be further exploited to answer RQ2 and RQ3 while aiming to meet RO3. For this purpose, we proposed an optimization algorithm that can train DNN in a lightweight and robust scenario. To assess its performance, we compared our optimized training procedure with the baseline DNN training method counterparts and other state-of-the-art conventional ML training methods for benchmarking purposes. Regarding robustness, the resulting training method can produce a well-resist DNN method that can defeat state-of-the-art perturbations. This is good as it can enhance the security of the proposed model to be a potential candidate for deployment in realistic IoT networks.

To further address AI based security solutions deployment, an RO4 is set up. The goal is to examine the feasibility of using DNN in a realistic resource-constrained decentralized environment. As a proof of concept, the evaluation outcomes aim to demonstrate that the IoT security solutions proposed in this thesis can appropriately address resource limitations and security challenges in an IoT network environment. In the evaluation, four realistic GB-BXBT-2807 edge computing devices were used. In addition, both IoT and non-IoT datasets are utilized and tested against various DNN architectures. The experimental evaluation discussed in section 6.3 tests the generalization capability of DNN in providing effective and accurate performance in real-time.

1.3 Research Contributions

In the quest of addressing IoT security challenges in resource-constrained environments, this research made the following contributions.

In addressing the challenges of using ML in IoT security monitoring in terms of both resource limitations and accurate detection, an efficient boosting method for resource constrained IoT devices is developed. In order to demonstrate the concept, a LGBM model was used, and its less computationally expensive parameters were factored out to create its better counterpart. The optimized LGBM can accurately discern attacks and regular traffic as tested on various IoT datasets that capture realistic attacks and benign instances from the IoT network's environment. In addition, the optimized method demonstrates its performance in terms of better attack detection and resource minimization. To the best of our knowledge this is the first attempt to optimize LGBM technique to save training and testing computational resources using multiple IoT benchmark datasets.

In addressing the effectiveness of using DNN that outperforms ML technologies in an IoT environment, an exploration of the Fully Connected Neural Network (FCNN's) optimization algorithm is examined to obtain the Resource Efficient DNN (REDNN) version from the FCNN. The experimental results are promising, as the resulting REDNN maintains better classification performance, low execution time and memory consumption. The resource minimization at training and testing time against multiple bencmark dataset is important. To the best of our knowledge this is the first work that investigates the resource efficiency of FCNN model against its benchmark counterparts using multiple device-centric IoT datasets.

In addressing the robustness of using DNN, REDNN is investigated in a robust scenario using adversarial perturbations created from a large number of benchmark datasets generated by hostile attacks on commercial IoT devices. Experimental results showed that the resulting REDNN is well resistant to adversarial attacks against each dataset used in our experiments. To the best of our knowledge, this is the first attempt to examine FCNN's capabilities for resource-efficient and robust detection in the IoT security domain. In particular, the exploration of adversarial perturbations using low precision floating point in the context of IoT security monitoring.

By addressing the challenges of using DNN methods in a federated environment to provide effective security solutions on IoT networks, an effective training method for DNN for IoT security monitoring is proposed. This training procedure can reduce the memory footprint and execution time during the training process while maintaining the same or higher level of accuracy than its benchmark counterpart. This work has been extended to develop a suitable federated DNN-based method for the security monitoring of resource-constrained environments such as IoT in real-time. To the best of our knowledge this is the first work to investigate on-device learning in the context of security and resource efficiency using simulation and a realistic testbed setting with IoT benchmark datasets.

Contributions listed in this thesis are therefore novel and contribute to the body of knowledge in the field. Evaluation codes are publicly available for enhancement and reproduction purposes.

1.4 Thesis Structure

The rest of the thesis is structured as follows:

Chapter 2 presents a research background on AI from the IoT security perspective. It further presents a review of relevant AI-based IoT security solutions in the literature in consideration of their strengths and weakness. In particular, the research investigates the potentiality of using traditional ML and DNN for cyber security. Finally, the research exploits model regularization, micro-batching and federated learning intending to optimize the DNN method for effective and efficient IoT security monitoring in a resource-constrained environment.

Chapter 3 presents the technical aspect of using ML for IoT security monitoring purposes. It starts with the exploration of traditional ML techniques as well as their optimization to reduce computational complexity. It examines the application of lightweights ML techniques for IoT cyber security. The chapter first exploits the potentiality of feature reduction techniques in reducing model computational costs. It carefully utilized GI and PCA to train an AIS algorithm and record the resource-savings advantage. Experimental results show that integrating AIS with PCA data transformation is less computationally expensive than the conventional AIS counterparts. However, AIS is not a suitable ML algorithm for IoT security monitoring using multidimensional data features. Motivated by these, the research later investigates a more generic method of training an ML algorithm in a resource-constrained environment. It examines a more promising ML method (DTEM) against several IoT benchmark datasets to develop a suitable IoT security monitoring method. In each case, details about the algorithm used and data pre-processing steps applied to each model are explained.

Chapter 4 presents the application of DNN models and their performance capability over traditional ML techniques for accurate classification with multi-sensory and large dimensional datasets. It starts by examining an appropriate method to train a DNN model in a resource-efficient manner. Based on this, regularization, micro-batching or model parallelism are utilized in developing an optimized DNN training procedure. The overall goal is to build a lightweight training method for DNN to improve accurate IoT attack detection while saving significant memory resources.

Chapter 5 aims to develop a robust and effective security scheme for IoT devices. The evaluation investigated various state-of-the-art perturbations techniques while utilizing a large number of benchmark datasets generated by hostile attacks on commercial IoT devices.

Chapter 6 explores the potentiality of our proposals in this thesis in a real-world scenario. It first investigates training DNN in a resource-efficient federated manner using decentralized edge devices. The experimental evaluation utilized a Federated Averaging (FedAvg) DNN along with eight IoT benchmark datasets to build the proposed method. The experimental results are encouraging as the resulting technique shows lower memory consumption with better classification performance in simulated and real testbed federated settings against each dataset used in the experiments. In addition, the federated integration of the model also helps to preserve the privacy of IoT device data during on-device model training.

Chapter 7 summarizes the outcomes of each chapter while justifying the contributions discussed in this thesis. It later presents an overall summary of the thesis. It further highlights future research directions.

Chapter 2

Study Background and Literature Review

This chapter explores the IoT in the context of cyber security monitoring and resource constraints to identify recent security challenges. In particular, the various attacks on IoT sensor networks and their implications. It later presents state-of-the-art AI techniques utilised in this thesis to propose IoT security solutions. It further describes related studies by exploring the recent ML and DNN detection algorithms used for IoT security monitoring. Especially those used to address security and privacy issues while using AI-based solutions in resource-constraint environments. Then, it presents a taxonomy of these exploited techniques used in this thesis with a discussion of the state-of-the-art adversarial perturbations techniques used against AI. The chapter finally explores recent federated learning methods used in IoT environments intending to address data privacy, security, and resource efficiency in realistic decentralized IoT network environments.

2.1 IoT Environment

The IoT environment refers to connected physical devices that interact through various communication protocols such as Bluetooth, Zigbee, Z-Wave, and WiFi to carry out specific operations. The functionality of IoT devices to ease our daily tasks is diverse. This makes them acceptable technology with various applications in healthcare and many industries. For instance, IoT wearable devices can enable remote patient monitoring by physicians and allow people to monitor their health status intending to improve their well-being. IoT devices can support inventory management of patient prescriptions and medical instruments in hospitals. IoT devices can be used to monitor and analyse machinery in smart-based agriculture to observe their functionalities. Business organizations can use IoT devices to monitor products in real-time and identify defects and other related issues. Moreover, IoT devices can monitor and track trucks in real-time to report faults in connected logistics. This spectrum of IoT applications becomes possible due to the architectural support of the IoT network environment. A typical IoT architecture can be described in four levels at which data flow within a network. The first level contains the devices (sensors and actuators), the second level consists of the gateway and the third and fourth level consist of the fog and the cloud, respectively. Figure 2.1 shows these four levels of IoT architecture that devices used to communicate with each other at a different points. At the device level, sensors can generate data that emerge from actions within a process, such as motion detection, environmental temperature condition, and air humidity. Actuators are responsible for performing specific tasks depending on the data derived from sensors. In a scenario in which the movement of an industrial robot needs adjustment, they need to be integrated with resource-efficient responsive mechanisms. At the sensory level, it is possible to perform certain computations at the edge of the network. This can be processing the data from the various connected IoT edge devices in a decentralized manner [17]. This is to address the concern of data privacy, while performing real-time data processing. The internet gateway is known as the data acquisition system. It is responsible for receiving raw generated data from the sensory devices that can be transmitted within a network. This procedure is required to ensure the data is ready to be recorded before sending it to the next level for immediate analysis. The fog level serve as a mediator between the edge devices and the cloud for various data processing. In the cloud layer, data flows from different fog nodes to the cloud servers [18]. These IoT systems of connectivity can be further illustrated in a multi-layered architecture shown in Figure 2.2. These are the perception layer, transport layer, processing and application layer. The layer containing smart devices is called the perception layer due to their sensing capabilities. The processing layer is responsible for the management and accumulation of the data gathered from sensory devices. The connectivity layer enables the movement of data from the perception layer to the cloud via an internet gateway. At this layer, the connectivity between devices can either be via the Transmission Control Protocol (TCP) / Internet Protocol (IP) stack or the internet gateways. These are the common method of transmitting data from the connectivity layer to the cloud. With the employed mainstream IT devices, the cloud stage enables certain operations with the data, especially the data storage and analytics process.

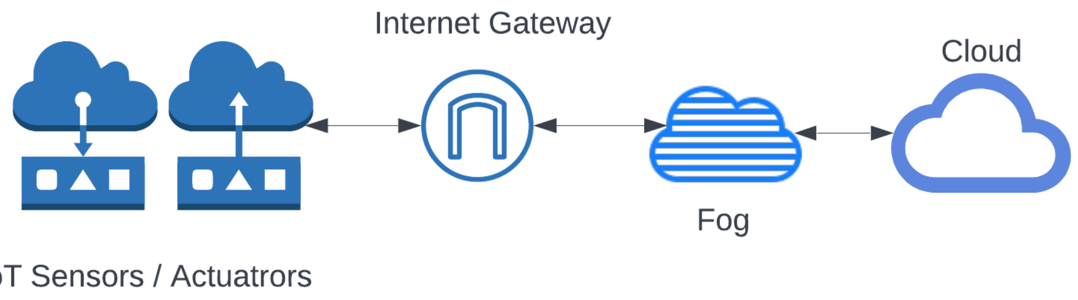

Figure 2.1: Four levels IoT architecture.

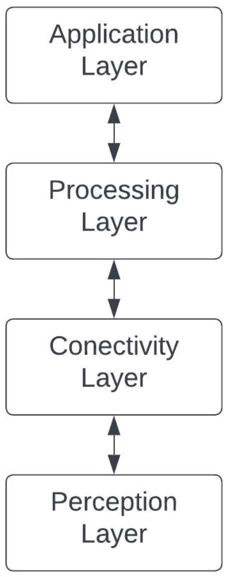

Figure 2.2: Four layers of an IoT architecture.

Unlike mainstream IT devices, IoT devices have limited memory, processor and computational power. With the focus of memory, most of them can only accommodate traditional flash and embedded flash memory [19]. The capacity of multiple-time programmable embedded memory is less than 256 KB, and that of the Atmel SAM R21 device memory can be up to 512 KB [20]. In addition, the Internet Engineering Task Force (IETF) categorized most IoT devices as constrained in nature using the RFC 7228 standard [21]. As the scope of this thesis considers the memory limitations aspect of IoT, Figure 2.3 shows the RAM and flash memory of different constrained IoT classes. Class 0 contains the most (ultra low) resource-constrained sensory devices with limited memory capacity much lower than 10KB for RAM and 100KB of flash memory. In this class, direct communication via the internet may not be possible, as the means of communication remain proxies and gateways. Class 1 devices contain limited storage space and processor capacity. Devices in this class are constrained in nature. In this class, communication between devices is not feasible via the Hyper Text Transfer Protocol (HTTP). The devices in class 2 have more storage capacity than those in class 0 and class 1. However, they require lightweight communication protocols and other resource-efficient schemes. For these reasons, IoT devices might not be able to execute multiple requests compared to mainstream IT devices without service disruptions. Therefore, they are becoming a potential target for cyber attacks. Unfortunately, the traditional AI-based security solutions or most antivirus software that require a certain amount of computational resources (memory and power) may not scale through IoT devices. Because of that, security solutions for IoT need to be resource-efficient and effective. In addition to the memory limitations of IoT devices, they also have a limited processor and computational power requirements. However, with the scope of this thesis, we focus on the memory limitations aspects of IoT devices.

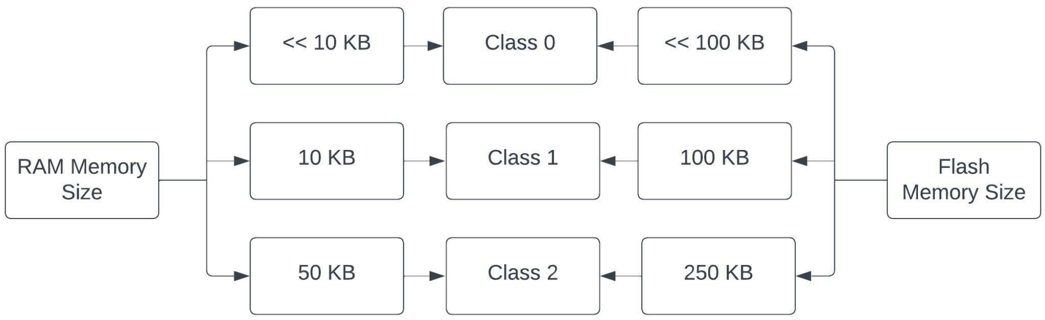

Figure 2.3: Constrained IoT classes memory requirements.

2.2 Attacks on IoT Devices

IoT technology is not yet mature and fully secured. Several security challenges occur, and vulnerability issues are key to them. Vulnerability attacks on IoT depend on the nature and actions of attackers. Some of these attacks are action-oriented. In that scenario, attackers act to control communications, acquire information, alter data, deny a service or damage network assets within devices. They take advantage of their resource-constrained nature in launching severe attacks. The description that follows discusses the commonly used such attacks. Most of them are present in the dataset utilised in this thesis to evaluate our proposed methods.

Scanning

Scanning attacks appear to be the initial and most popular cyber attack techniques in the IoT attack life cycle. In this attack, an attacker utilised scanner IPs from compromised IoT devices to probe available open ports in the network. By executing this attack, other limited resources of the compromised IoT devices can be exhausted. By using a specific device, this attack can be launched using popular scanning tools such as Masscan, Zmap and Nmap to prove a million packets in a network within a second [22].

Denial of Service (DoS)

DoS is determined to disrupt a device intending to alter its functionality within a network. The procedure involves exploiting the device with a vulnerability or flooding the device network traffic [23]. In each case, the intention is to slow down the device or destabilize its operational capability. A DoS attack can be software or hardware-based and may involve alteration and destruction of sensitive information. With the syn flooding procedure, a DoS attack can exhaust resources from the targeting devices or remote servers. This attack can be distributed in nature as in (DDoS) or denying permanent access (PDoS) [24].

Reconnaissance

In this attack, the adversary aims to discover the knowledge of specific IoT networks by sending internet information queries to various devices. Reconnaissance attacks include traffic analysis, port scanning, and packet sniffing [25]. It consists of the set of tools that the attacker utilised to gather legitimate information about a specific target. The attacker can use social engineering methods to obtain information about a particular company from social networking websites. In this case, the attacker can access a company business model hosted over the internet. In most cases, the attacking method remains a digital process without physical human intervention.

Spoofing

This attacking procedure allows the adversary to act as a legitimate source in a network. In this case, a compromised device broadcasts the shortest route messages within the network. As such, the attacker can send vulnerable messages to compromise IoT devices [26]. A successful spoofing attack enables the cyber criminals to obtain access to sensitive resources in a network intending to inject malicious content into the system. The most common type of this attack is email spoofing which can be carried out using social engineering techniques. Other target victims of spoofing attacks include websites, IP addresses, Domain Name System (DNS), Address Resolution Protocol (ARP), and Global Positioning System (GPS).

Sybil

In Sybil attack, an adversary can have multiple fake identities within a network. Sybil attack is determined to breach the security of the data and reduce the resource capacity of a device [27]. It can reduce the overall performance of IoT devices within a network by allowing a single device to perform multiple tasks. Therefore, this type of attack requires effective and efficient countermeasures.

Sinkhole

In this attack, the compromised device advertises a broadcast message claiming to have the shortest path to the base station within the network. This is determined to attract the traffic of the nearest sensory nodes into a single node. This attack degrades network performance while collecting legitimate information. It can consequently harm other IoT devices from the base station. It can utilise the selective forwarding attacking technique to alter the sensory data collected by communicating IoT devices in a network [28].

Message Injection

Message injection involves capturing a message within a network and altering its contents. In this attack, the attacker tends to send falsified information to perform malicious activities, such as corrupting records, gaining unauthorized access or simply overloading the network with compromised requests. This attack can overwhelmingly affect the sensory devices and prevent them from delivering their intended messages while disclosing their integrity in the communication process. In addition, cyber criminals used this type of attack to inject a message into connected vehicles in a Controller Area Network (CAN) to take active control [29].

Command Injection

This is the most common and dangerous attack launched by the cyber criminals to compromise IoT devices. This attack target a specific communication interface used by IoT devices to inject malicious commands into those devices. In this attack, an attacker can execute commands to exploit, gain access to many IoT devices to upload malicious content, change their network configuration settings and obtain passwords. With this attack, a bot can be created and installed into an IoT device such as an IP camera, webcam and many household smart appliances to compromise other devices in a network [30]. This attack is severe and requires efficient and effective detection mechanisms.

Backdoor

In a backdoor attack, the attacker observes the weakest part of the embedded software of connected IoT devices to infiltrate the system maliciously. This attack utilised malicious programs such as Trojans to launch sophisticated attacks on IoT devices. It is determined to gain access to and control the communication between various IoT devices remotely in a network. In another scenario, it can be the connectivity point for ransomware, Trojans, viruses and spyware that can penetrate to a network to cause cyber disaster [31]. Attackers used it frequently to breach the security of IoT devices. Therefore, backdoor attacks remain critical threats in the IoT security landscape.

Keylogging

In a keylogging attack, an attacker can install a malicious log file (keylogger) to read and steal the encryption keys present in a device. It is the oldest form of stealing passwords, reading logs and keystrokes and recognizing patterns within an existing system or devices to install bugs and other malicious contents. With this attack, cyber criminals can monitor your devices and gain unauthorized access to personal and legitimate information, record activities and steal sensitive data [32].

Data Exfiltration

In this attack, an attacker intends to steal or cease the data transmission process within devices without permission. Data exfiltration attacks target sensitive information from organizations to cause data loss, destruction, and leakage while degrading the functionality of the victim's organization. It first started by exploring the target organization to gather necessary information, mostly weaknesses of an exiting system that can make the attack feasible [33]. A successful data exfiltration attack can steal user credentials stored on various IoT devices or make an existing network inactive for many hours. As reported by the Identity Theft Resource Center (IETF) 2021, there are about 1603 data compromised incidents reported with a 495 increase in incidence rate than 2020 [34]. With this interpretation, this type of attack requires effective countermeasures mechanisms.

As ML techniques offer promise for various classification tasks, they can be the potential scheme for detecting many cyber attacks. By exploring and improving the available AI-based security mechanisms utilised for cyber security monitoring, effective and efficient security solutions can be developed and deployed in resource-constrained environments. Table 2.1 presents a summary of the procedure and impacts of various security attacks on IoT devices discussed in this section. The illustrated attacks are captured in generating the various publicly benchmarked IoT datasets described in section 2.3.9 and utilised in this thesis.

Table 2.1: Attacks on IoT

Attack	**Procedure**	**Impacts**	**Target**
Scan	IoT device port and IP scanning.	IoT device resource exhaustion.	High profile organizations.
DoS	IoT device traffic overloaded.	IoT device service disruption.	High profile organizations.
Reconnaissance	Scanning / analyzing IoT device network traffic.	Accessing sensitive IoT device location information.	Online social networking site.
Spoofing	Sending vulnerable queries to IoT devices.	Service disconnection for IoT device.	High profile business company.
Sybil	Creation of vulnerable IoT devices.	Breaching IoT device data integrity.	Online social networking site.
Sinkhole	Creation of compromised IoT devices.	Degrade IoT network service performance.	High profile organizations.
Message Injection	Injecting malicious content to IoT devices.	Disclose IoT device data integrity	High profile organizations database.
Command Injection	Malicious commands injection to IoT devices.	Threats to IoT device gateways.	High profile company websites.
Backdoor	Infiltrate IoT devices maliciously.	Breach IoT device network interface.	High profile organizations.
Keylogging	Installing logs and keystrokes IoT devices.	Accessing and monitoring IoT devices.	High profile organizations.
Data Exfiltration	Exploiting IoT devices for victimization purposes.	Degrade IoT device data collection capability.	High profile organizations.

2.3 AI Techniques for IoT Security Monitoring

As a subfield of AI, ML enable machines to learn without explicit programming. ML learning algorithms require a set of input samples for training to learn the relationship between training inputs and training targets (see Figure 2.4). ML can distinguish patterns and predict labels for unseen data during classification. This is the supervised form of learning which focuses on predicting the appropriate labels for an unseen feature vector by learning how to map the input feature data to the desired target. By given a set of training examples X = $\{(x_i, y_i)\}$, $i = 1 \ldots t$ containing feature vector (x_i, y_i) in discrete, continuous or numeric forms, ML algorithms can outputs a classifier that can assign desired labels to each feature vectors. In this case, ML improves in a programmable manner by learning the pattern from the data. This form of learning is the most commonly used in speech recognition, image classification, language translation, anomaly detection and attack identification. In the unsupervised training scenario, ML can learn the semantic distribution of a given vector to get an insight into its properties. This form of learning is determined to find an interesting pattern present in the input data without using the target. In addition, ML can be trained based on rewarding desires or undesired actions from its environment in a reinforcement manner [35]. These AI-based (ML) concepts are widely used to address various issues in different domains. However, this thesis considers the supervised version of ML, as most of the IoT network traffic benchmark datasets contain labels for benign and attack samples. Therefore, supervised learning can be an appropriate mechanism that can identify attacks and benign traffic samples captured for cyber security monitoring tasks. Within this context, ML techniques can be utilised and improved to examine and detect attacks on an IoT network environment for security monitoring purposes. Figure 2.5 shows the relationship between AI, ML and DL (DNN). In this chapter, 112 research works are examined from ACM, Scopus, Google Scholar, IEEE Xplore, as well as other online resources by using keywords such as IoT security, effective IoT security, machine learning for IoT security, deep learning for IoT security, intrusion detection in IoT networks, on-device learning, and adversarial attacks. Most of these papers are review in section 2.4. In addition, most of the review papers in section 2.4 are from 2016 to 2022. The detail of the AI-based ML and DNN algorithms utilised in this thesis are as follows:

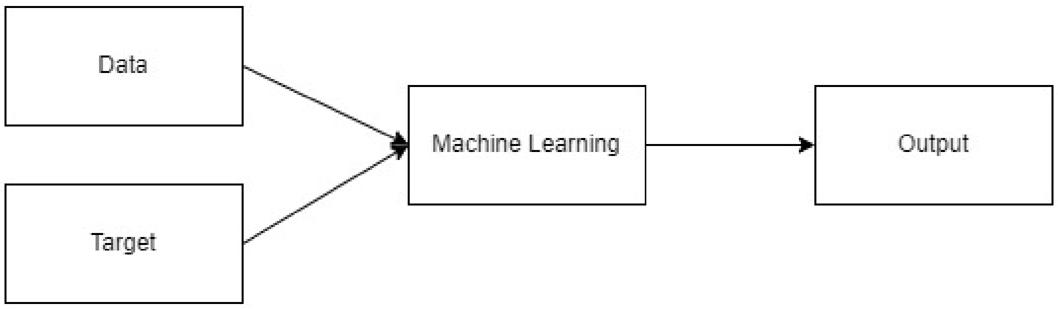

Figure 2.4: An illustration of a supervised ML.

2.3.1 Decision Tree Ensemble Methods (DTEM)

With DTEM, multiple learners can be formulated to produce a better predictive-based classification model by a procedure called an ensemble. DTEM ensemble procedures are categorised into bagging and boosting. Bagging is an ensemble decision tree algorithm that manipulates the training data instances to improve classification model performance

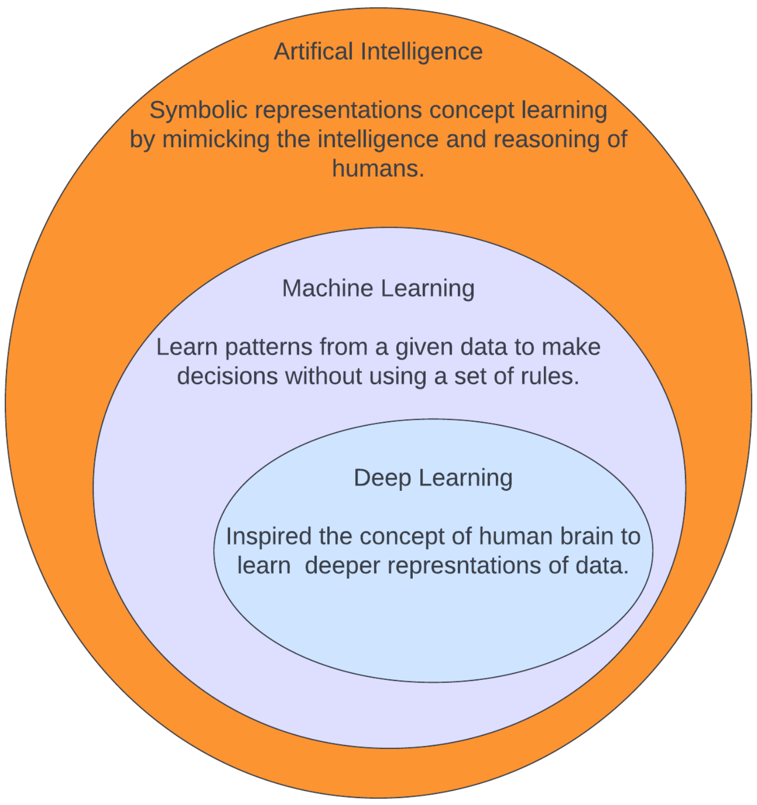
Artifical Intelligence
Symbolic representations concept learning by mimicking the intelligence and reasoning of humans.
Machine Learning
Learn patterns from a given data to make decisions without using a set of rules.
Deep Learning
Inspired the concept of human brain to learn deeper represntations of data.

Figure 2.5: Relationship between AI, ML and DNN.

[36]. This involves a random selection of data samples with a replacement by which specific data points can be chosen multiple times. This is useful in training the weaker classifier models independently while applying voting of their predictions to find an accurate output. Boosting is an ensemble method that involves the reduction of the classification error generated from the previous classifier. It is a way of training a new ML model to improve its performance so that it can be an appropriate candidate to address the weakness of an existing (previous) model. This technique is more sensitive to model over-fitting on noisy data with extensive training iteration [37], and it can also be referred to as Gradient Boosting Decision Tree (GBDT). Because of these ensemble procedures, DTEM has multiple variants. For clarity purposes, the following discusses the various DTEMs utilised in this thesis. These DTEM variants are Random Forests (RF), LogitBoost, Adaptive Boosting (AdaBoost), and LGBM.

(i) Random Forest (RF)

RF is a decision tree method that can train multiples tree to generate output that can decide the classification outcome of a given data instance. At training, each tree utilises a random subset of the training samples [38]. In classification, the output of each tree is essential to determine the class label of an unseen feature vector. Given a training set R = $\{(r_i, c_i)\ \}$, $i = 1 \dots l$ with data instances r_i and class labels c_i, RF applies bagging to the training data and uses voting to predicts the classes of unseen data instances. An advantage of RF is the accurate or better prediction of unseen data instances [39]. Figure 2.6 illustrates a simplified diagram of a random forest classifier. It consists of input data instance and a forest of n trees with their respective decision based on the given data instance. With this procedure, these trees learned from the data instance and produce their output. During the decision analysis, voting is applied based on the decision outcome in an ensemble manner at which classifiers with better prediction scores are considered.

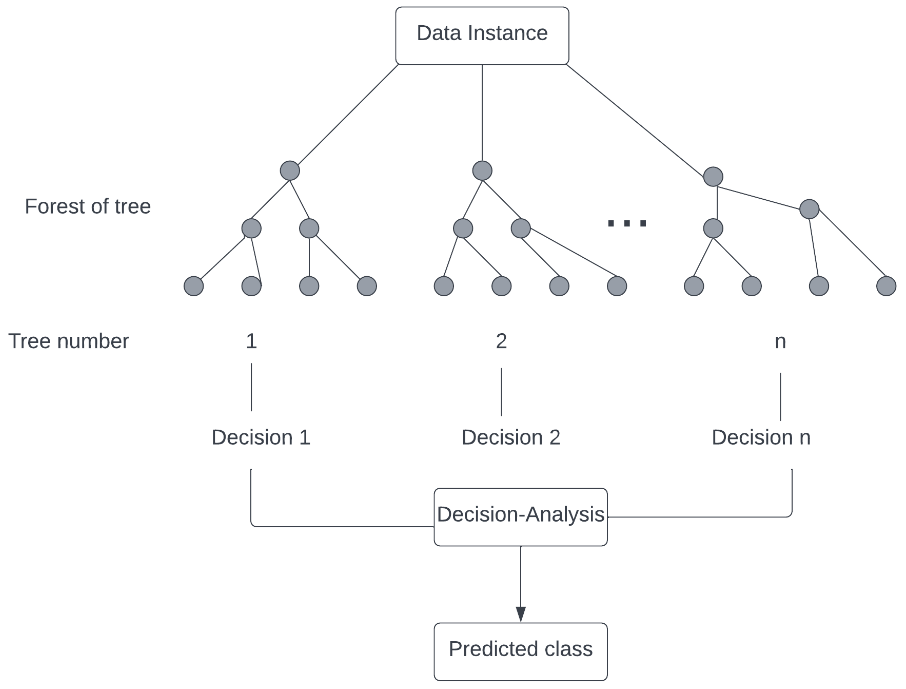

Figure 2.6: An illustrative diagram of a random forest classifier.

(ii) Adaptive Boosting (Adaboost)

Adaboost is an ensemble method that creates an initial classifier using a given dataset. By manipulating the given training dataset, it can produce multiple copies of that classifier. In each iteration n, weights $w_j, j = 1 \dots n$ are computed and assigned to the classifiers based on their prediction performance. The weaker classifiers take the higher weights values [40]. An Adaboost algorithm is determined to minimise the predictive error function from multiple classifiers by voting on their performance. This can be defined using a predefined threshold t that serves as the margin of a classifier upon predicting a particular label correctly or otherwise. The score value associated with each classifier determines its weakness, where a higher value means a large significant error and weak performance. With an assigned label of 1 for benign instances and 0 for attack instances, then t_i takes the value 1 for the correct classification of benign instances and 0 otherwise. An Adaboost model that tries to minimises the logistic regression loss function in 2.1 is called a Logitboost [41]. Where D is the dataset containing the input samples X and the target Y, $\hat{Y}$ is the predicted value, given the set of features vector X. An Adaboost classifier is capable of performing accurate classifications better than other conventional ML models [42]. We utilised this technique to benchmark our proposed optimised models in chapter 3.

$$Logloss = \sum_{(X,Y)\in(D)} -(Y \log \hat{Y} + (1-Y) \log (1-\hat{Y}) \qquad (2.1)$$

(iii) Gini Index (GI)

GI is an inductive decision tree algorithm based on an impurity function, called *gini index*, for finding the best split. The impurity function can measure the feature importance of a given dataset. In particular, to indicate the likelihood of misclassifying new data based on the random distribution of its classes [43]. GI method explores the relative distribution of a feature among classes and is a useful resource reduction method. This technique was developed by an Italian sociologist and statistician called *Corrado Gini* in 1912. The main idea is to measure the statistical dispersion of income across various populations. The method has a wider application in IoT research to purify important features [44]. The *Gini, G* impurity of a dataset S having m subset $S = \{s_1, s_2, s_3, ..., s_m\}$ with j different subclasses $C = \{c_1, c_2, c_3, ..., c_j\}$, is defined in Equation 2.2.

$$G(S) = 1 - \sum_{j=1}^{m} P_j^2 \tag{2.2}$$

Where, P_j is the rate of class c_j in S; S can be split into m subsets, m_i is the sise of subset S_i as described in Equation 2.3. The split with the best value (smaller impurity) among classes is chosen. This is because the smaller the impurity value, the better the splitting node [45]. This process is referred to as the feature impurity gain score. The range of the splits of $G_{split}(S)$ is between [0, 1].

$$G_{split}(S) = \sum_{i=1}^{m} \frac{m_i}{m} Gini(S_i) \tag{2.3}$$

2.3.2 Light Gradient Boosting Machine (LGBM)

LGBM is a decision tree algorithm based on gradient sampling of data instances with smaller gradients and exclusive feature bundling [46]. The gradient sampling method suggests a specific search space rather than an entire search space [46]. LGBM uses the information gain ranking in the gradient sampling process. A smaller gradient value indicates a well-trained data instance which needs to be separated from those with larger gradient values. As such, LGBM discards those data instances with the larger gradient and performs one-sided sampling on data instances with a smaller gradient. As a result, the technique can converge faster and stands more scalable than most decision tree counterparts as it does not search through the whole searching space. A limitation of LGBM is the requirement of various parameter tuning in the gradient selection procedure as it uses the default boosting parameters of GBDT [47]. In addition, it requires a specific architecture for the task of regression, binary, and multiclass classification. These are challenging tasks using multidimensional feature data. In particular, with the IoT device-centric data collected from many commercial devices. Therefore, LGBM requires effective optimisation for appropriate deployment in IoT resource-constrained environments. Following this discussion, a novel LGBM training algorithm is proposed in section 3.2.1 to resolve these limitations.

LGBM Hyperparameters

An LGBM learning model has various hyperparameters to consider for optimisation phases and implementation, particularly for the task of binary classification. These include the number of leaves, feature fraction, bagging fraction, bagging frequency, learning rate, and a regularisation term. Table 2.2 described the range of values of these hyperparameters that can be tuned to produce a better classifier. There is a recommendation for varying each hyperparameter value as described in the LGBM module [48]. Feature and bagging fractions values must be set within [0, 1] depending on the employed dataset and the task to accomplish [48]. The feature fraction parameter value represents the size of the selected feature vector subset from the training data at each iteration. Setting the feature fraction to 0.2 at training means 20% of the data features will be used to train the LGBM model. A smaller feature fraction value can speed up the training procedure. In addition, enabling bagging with its frequency set to a non-zero value can facilitate efficient model learning. The regularization alpha value as a constraint parameter can be greater than 0.0 for better model fitting and faster learning. It works perfectly with datasets with larger than 10000 samples [48]. A smaller number of leaves can avoid model over-fitting while a larger value can increase the accuracy performance, especially with large training data. This depends on the task to accomplish and the nature and complexity of the training data. The learning rate chosen value depends on the number of iterations and training data size. Tuning the learning rate parameter helps in balancing the trade-off between training computational expensiveness, the chance of model over-fitting, and performance accuracy [49]. These parameters are essential in exploiting the LGBM to reduce its computational complexity and improve its performance for cyber security monitoring because the default parameter values may hurt the model accuracy [49]. Based on these hyperparameters, we trained LGBM using our proposed optimization procedure to produce its resource-efficient counterparts for the security monitoring of resource-constrained IoT devices (see section 3.2.3).

Following the discussion of the DTEM, Figure 2.7 depicted the root and branches for the ensemble DT method. For the task of binary classification with two data instances (X1 and X2). For classifying XN data instances, several tree nodes can be used iteratively.

Table 2.2: LGBM hyperparameters.

Hyperparameter	Minimum	Maximum
Bagging Fraction	0.0	1
Feature Fraction	0.0	1
Number of Leaves	1	131072
Learning Rate	0.0	0.1
Regularization Term	0.0	0.1

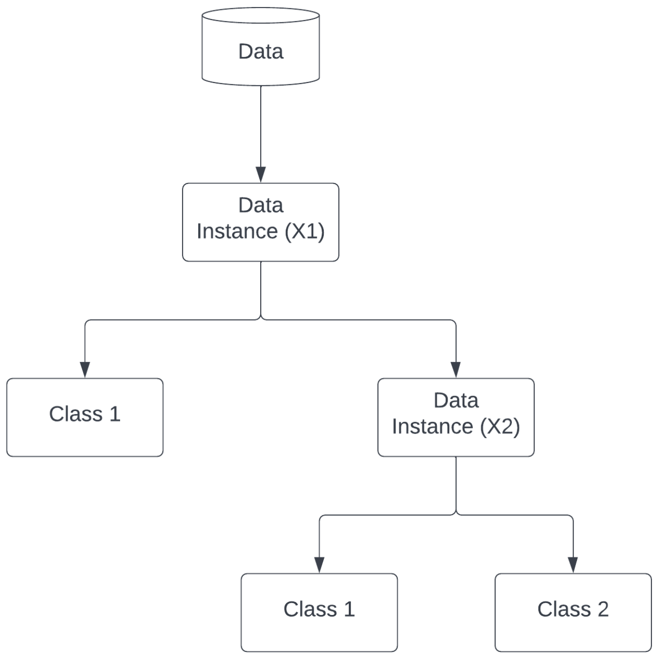

Figure 2.7: Component of decision tree for binary classification.

2.3.3 Support Vector Machine (SVM)

SVM is a supervised ML algorithm that creates a hyperplane or decision boundary to separate data instances into classes [50]. For the task of binary classification, SVM intends to find the hyperplane with the maximum margins (see Figure 2.8). An ideal hyperplane can differentiate the two classes. In Figure 2.8, it separates the red and blue classes, respectively. A margin is the distance between the data points and the dividing line. By considering the maximum distance between the data instance within the two classes, an SVM classifier tries to maximized this margin to find an optimal hyperplane. For 2-dimensional data, the hyperplane is 1 dimension while for n-dimensional data, the hyperplane is n-1 dimension [50]. The support vectors in Figure 2.8 are the closest data points to the hyperplane. SVM used kernel functions to project data into a higher dimensional space to influence better learning. A kernel function can be a linear, polynomial, radial basis or sigmoid [51]. The kernel function served as a trick to manipulate a given training data. In the case of the linear version, it can find the linearly separated patterns present in the data. For implementation purposes, different values can be specified using the *kernel* parameter. Figure 2.8 illustrates a two-class SVM classifier. An SVM classifier can be utilised to detect anomalies in the context of cyber security monitoring [52].

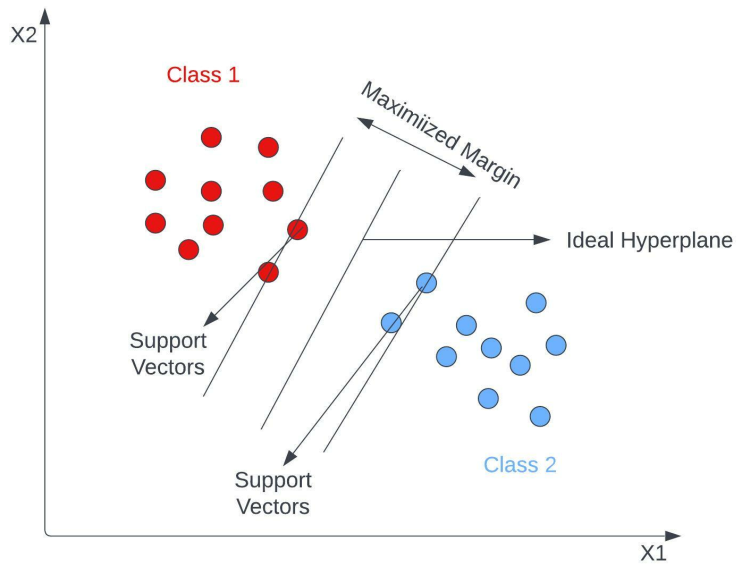

Figure 2.8: An illustration of two class SVM.

2.3.4 Artificial Immune System (AIS)

Computer scientists are motivated by the biological systems in developing techniques for solving problems, based on the idea that an immune system can be used to classify data instances. They utilised the Negative Selection Algorithm (NSA) from an AIS for classification [53] and cyber security monitoring tasks [54]. With its capability for recognizing patterns, this algorithm trains a population of antibodies called detectors using a benign sample from a given labelled dataset. The first step for training NSA algorithm involves defining two different datasets, the original input data and the randomly generated artificial data based on the size of the original data. A Real Value NSA (RNSA) generates random detectors and tests them against the instances of the self-class for affinity measure. Affinity is measured based on distances such as Euclidean, Manhattan, or Cosine. If the affinity measured between the generated detectors and self class instances is above certain threshold, then it is not considered [55]. At implementation, there is no perfect shape for an antibody representation. However, RNSA utilised a hypersphere antibody. In training, the RNSA algorithm uses a realistic dataset to view every feature vector within the shape search space. This makes it easier to normalize the values in the data within the range of [0, 1]. Each feature vector is now associated with a point in the shape space. In the case of the RNSA algorithm that handles numerical data, the shape space (as well as the feature vector values) are continuous. Equation 2.4 express the RNSA.

$$A = R^d \tag{2.4}$$

The expression $A = \{a_1, a_2, a_3, ..., a_d\}$ represents data instances, R is the real value representation of data in the search space, d is the number of dimensions. This expression can be utilised in a dataset with $C = \{c_1, c_2, c_3, ..., c_n\}$ representative of the n class labels of a given data samples. In chapter 3, we trained an RNSA algorithm to investigate its resource consumption while integrating feature reduction techniques for IoT cyber security monitoring.

2.3.5 Principal Component Analysis (PCA)

PCA, known as the *Karhunen-Loeve*, is a statistical ML procedure that transforms an observed set of possibly correlated variables into a set of values of linearly uncorrelated variables, called principal components. The number of decomposed principal components is fewer than, or equal to, the original number of variables. The rationale for PCA is to identify the search space for analyzing multi-dimensional data samples in a low-dimensionality. For instance, an n dimensional data instance might be confined into an $n - 1$ distinct principal components. Hoang et al. [56] utilised PCA using a substantial data sample for IoT attacks detection. Figure 2.9 illustrated the dimensions of PCA with two principal components, the PCA-1 and PCA-2, respectively.

2.3.6 Neural Networks (NNs)

Neural Networks (NNs) or Artificial Neural Networks (ANNs) mimic the notion of the human brain to build computer programs that can recognize patterns and solve a plethora of problems. ANNs are the core concept of DNN that simulate the functions and structure of the human brain. They can be classified into different variants depending on the purpose. These are the perceptrons, feedforward neural network, convolutional neural network (CNN) and recurrent neural network (RNN). These ANNs are widely used to build AI-based solutions for various real-world problems [57]. The detection procedure can be supervised [58] or unsupervised [59]. Figure 2.10 illustrated the architecture of a shallow ANN, with one hidden layer. An approach that can compress DNN model architectures for faster processing and inference acceleration is called *Quantization*. It is

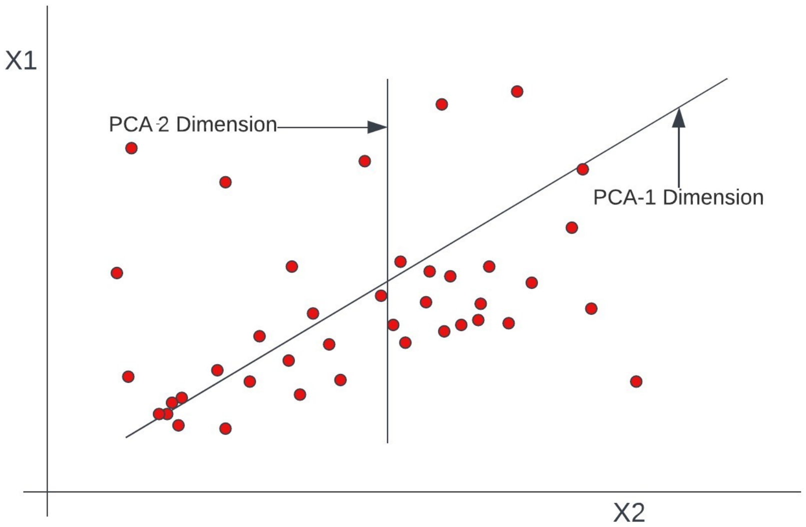

Figure 2.9: An architecture of PCA.

the procedure of mapping complex DNN architecture into smaller architectural representations. Liang et al. [60] outlines the various schemes of quantizing a DNN model. With quantization, specific operations in the network can be computed in different data types such as float 16 bit, float 32 bit or different integer precision. This low precision scheme is becoming the de facto training technique that can increase the energy efficiency of DNN hardware. However, they can harm DNN models by reducing their classification accuracy [61]. In section 4.3 of this thesis, we compared our training procedure for DNN technique with TensorFlow Lite that utilised low precision in model training to assess the effectiveness and resource efficiency of our optimised DNN training method.

2.3.7 Fully Connected Neural Network (FCNN)

FCNN is a neural network structured into several layers of neurons representing the input data. A neuron is a fundamental computing unit capable of transmitting the result of the operation computed by its activation function with the input. Each neuron represents an individual node within the network. They can connect in a non-linear pattern of layers containing activation functions to transmit data. A typical FCNN can connect neurons sequentially by linking them with their corresponding weights and bias parameters. These weights and biases function as information storage units. They are the threshold that controls the flow of operation computed by each neuron within the network. In this aspect, each node can utilise relevant features from the previous layers

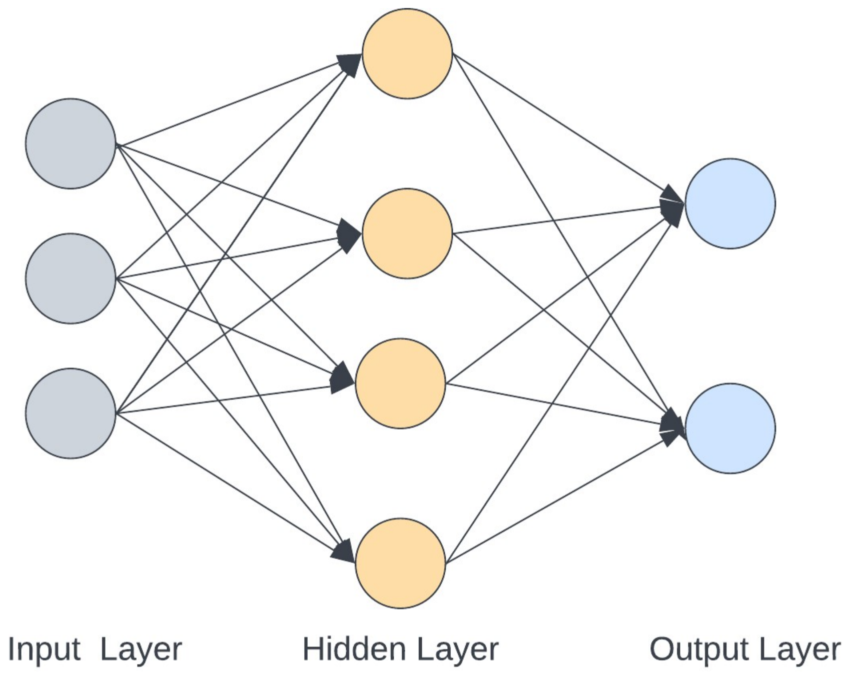

Figure 2.10: Typical ANN architecture.

while transmitting their output values to the next layer.

Figure 2.11 illustrates an FCNN architecture with two hidden layers and their respective neurons connection. From the illustration in Figure 2.11, the FCNN deep neural network can take three input feature vectors $\{X_1, X_2, X_3\}$ representing the units in the input layer. As indicated with the forward arrows, each feature unit of the input layer feeds into the first four hidden neurons, subsequently to the next three hidden neurons up to the last two output neurons. This unidirectional forward flow of the input features is derived by an activation function in a pattern called forward propagation. The forward propagation is responsible for organizing the collected intermediate neuron's input features. With N neurons in the hidden layers, $A_n, n = 1, ..., N$, activation functions are computed against the input feature instances. The most commonly used activation function in the hidden layers is the rectified linear unit (ReLU), while the sigmoid function is for the output layer. The ReLU function in Equation 2.5 is determined to be more computationally efficient in terms of computing and storing several activation functions within the network layers. It can be an appropriate feature transformation function in a network. The sigmoidal function expressed in Equation 2.6 can convert a linear function into probabilities between zero and one. It accommodates those values of hidden activation functions that are closer to one while neglecting those closer to zero. The learning procedure of DNN requires input data to predict a class label. This is important in training DNNs for a specific task. Especially, in classifying various data instances accordingly. To achieve this, DNN used algorithms such as gradient descent and backpropagation. The backpropagation algorithm is determined to compute the derivative of the objective function for the input data in a backward pattern. As such, it can propagate the computed gradients from the output node back to the input node within the network. This process is repeated over time until the network reaches a convergence point. To get insight concerning the DNN training procedure, see Algorithm 1. It is used to train and obtain an FCNN model ($\mathcal{M}_b$) as the core baseline utilised in chapters 4 and 5 in this thesis. The function BASE in line 1 of Algorithm 6 corresponds to the $\mathcal{M}_b$ mini-batch training using the gradient descent algorithm while computing the forward and backward derivatives [62]. The procedure is determined to minimize the objective function $J(W, b)$ (of weight W and bias b) in Equation 2.7, especially in minimizing the negative log-likelihood (cross-entropy) to map unseen samples by using a procedure that learned from $\mathcal{S}_{tr}$. The resulting FCNN approach uses supervised neural networks as a classifier, $\mathcal{M}_b$ can accept an input $\mathcal{S}_{tr}$ and outputs a probability class of vector $\hat{Y}$. The desired output $\hat{Y}$ are rounded up to the closest integer using a specified threshold value t as in Equation 2.8. This output represents either the benign (1) or the attack (0) traffic instance.

$$relu(x) = (x)_+ = \begin{cases} 0 \text{ if } x < 0 \\ r \text{ if } x \geq 0 \end{cases} \tag{2.5}$$

$$sigmoid(x) = \frac{e^x}{1+e^x} \implies \frac{1}{1+e^{-x}} \tag{2.6}$$

$$J(W, b) = \frac{1}{m}\sum_{i=1}^{m} L - (Y \log \hat{Y} + (1-Y) \log (1-\hat{Y}) \tag{2.7}$$

$$Detection = \begin{cases} 0 \text{ if } \hat{Y} \leq t \\ 1 \text{ if } \hat{Y} > t \end{cases} \tag{2.8}$$

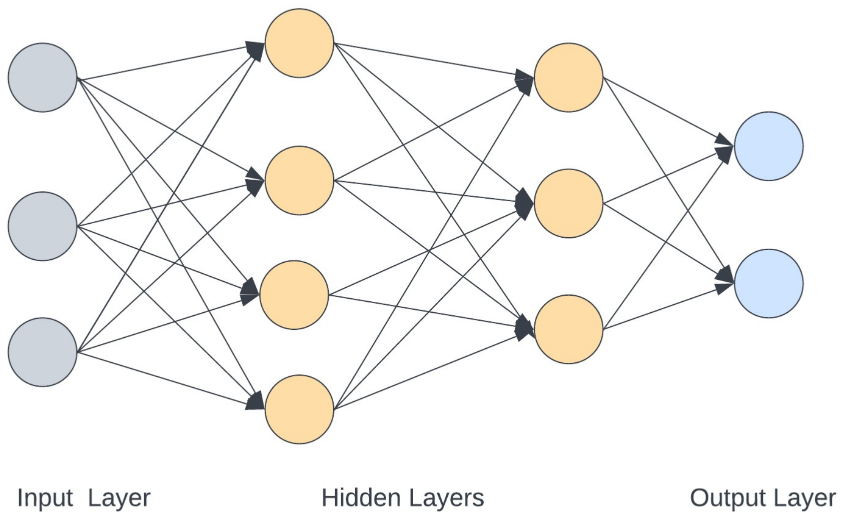

Figure 2.11: Fully connected (deep) neural network architecture.

2.3.8 Convolutional Neural Network (CNN)

CNN (ConvNet) is a special DNN that evolve to analyze and classify image data. CNN has been widely utilised to address various image-related tasks [63]. A typical CNN consists of neurons, convolutional layers, and fully connected layers. The fully connected layers are trained based on weight and biases parameters. In addition, CNN used various non-linear activation functions in the convolutional layers to process high-resolution images. In this context, the network can identify low-level features of the input image that are subsequently combined to form high-level features. The pooling layers of CNN are responsible for selecting a subset from the input image data. They can condense bigger images into a smaller version via max pooling. In CNN architecture, each convolutional filter can produce a new-multidimensional feature map. At training, convolution feature filtrations and max-pooling are repeated iteratively until the network converges. In this aspect, the CNN can actively learn from the input data for the task of accurate classifications. At this stage, the network can transform the input x using the softmax function in Equation 2.9 to output a probability distribution in the output layer that represents a class label C. In Figure 2.12, we illustrate the ML and DNN techniques investigated in this thesis for the proposal of effective and efficient IoT security solutions. Some of them are used to compare our proposed optimised training method. These includes, RF, AdaBoost, logitBoost, GBDT, LGBM, SVM. The PCA and GI utilization is to investigate the effectiveness of integrating feature reduction with AIS algorithms for IoT cyber security.

Algorithm 1 Baseline FCNN

```
    Input: Labelled data $\mathcal{S}_{tr}$, Number of iteration $\mathcal{K}$, Batch size $\mathcal{H}$,
    Output: Baseline model $\mathcal{M}_b$
 1: function BASE($\mathcal{S}_{tr}[\,]$)                                    ▷ Training baseline model
 2:     for $i = 1$ to $\mathcal{K}$; do
 3:         Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m)\} \subset S_{tr}$    ▷ Mini-batch size $\leftarrow |S_{tr}|//\mathcal{H}$
 4:         $F_i(B)$                                                      ▷ Forward propagation
 5:         $\mathcal{J}_i \leftarrow L$                                          ▷ $L$ = Base loss
 6:         $B_i$(B)                                                      ▷ Backward propagation
 7:         Compute gradients for parameters update
 8:         $\mathcal{M}_b$ = Trained model using mini-batch gradient
 9:     end for
10:     return ($\mathcal{M}_b$)
11: end function
```

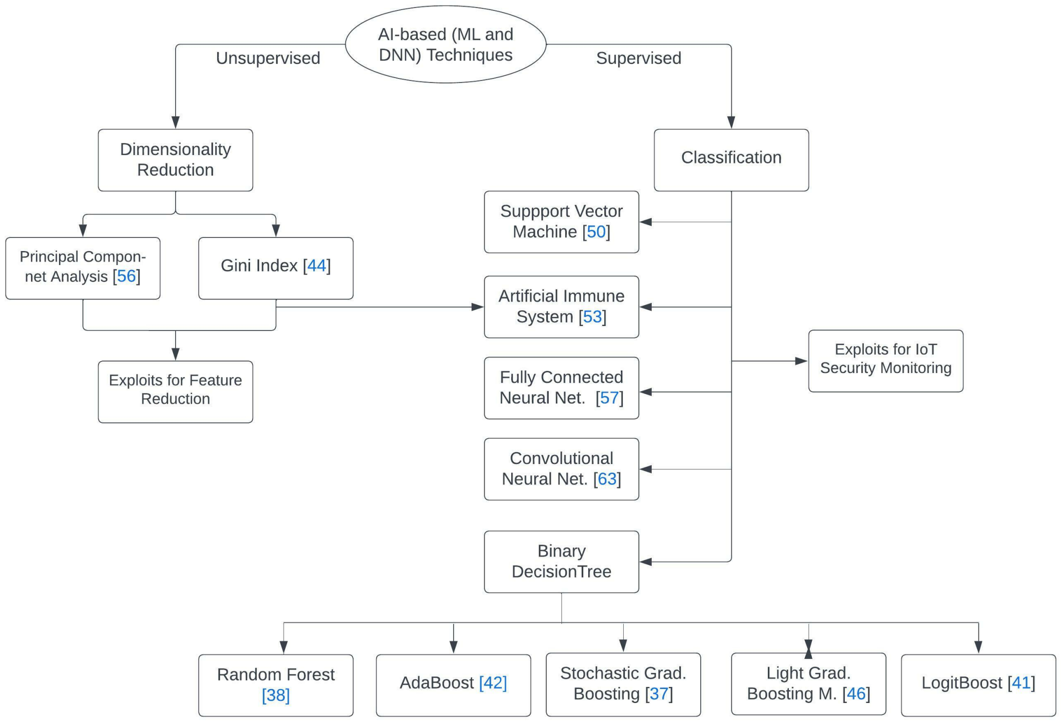
AI-based (ML and DNN) Techniques
Unsupervised
Supervised
Dimensionality Reduction
Classification
Principal Compon-net Analysis [56]
Gini Index [44]
Exploits for Feature Reduction
Suppport Vector Machine [50]
Artificial Immune System [53]
Fully Connected Neural Net. [57]
Convolutional Neural Net. [63]
Binary DecisionTree
Exploits for IoT Security Monitoring
Random Forest [38]
AdaBoost [42]
Stochastic Grad. Boosting [37]
Light Grad. Boosting M. [46]
LogitBoost [41]

Figure 2.12: Taxonomy of AI techniques utilised for IoT security monitoring.

$$softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{2.9}$$

.

2.3.9 Datasets

Benchmark datasets are essential for assessing the performance of ML techniques. In the context of cyber security of monitoring, several public network traffic datasets are available for emprical model evaluation.

(i) KDD 99

The KDD 99 cyber security dataset [64] is generated based on the modification of the Defence Advanced Research Project Agency (DARPA) [65] network traces which results in 41 connection of features together with the class label. The dataset consist of DoS, Probe, U2R, and R2L four category of attack records. After its creation, the dataset can be used to build ML techniques to distinguish attack and normal network traffic instances. The dataset is partitioned into training and testing data and is publicly accessible in the ML repository [66].

(ii) Unsw

The Unsw dataset [67] was created using the IXIA tool testbed configuration that generates attack and normal traffic behaviours. This dataset consists of nine simulated attack activities. These are the (i) Fuzzers, (ii) Analysis, (iii) Backdoors, (iv) DoS, (v) Exploits, (vi) Generic, (vii) Reconnaissance, (viii) Shellcode, and (ix) Worms [67]. The dataset consists of 48 features of these attack categories and 1 feature for a class label normal or attack. A partition of its training and testing data sample is made publicly available [68]. Based on the characteristics of its traffic features, the dataset was used to develop an IoT intrusion detection system [69].

(iii) N-BaIoT

The N-BaIoT dataset contains various realistic data samples from nine commercial IoT devices that collectively represent multitudes of botnet and benign network traffic flow [70]. Each device is either infected by BASHLITE or Mirai severe attacks, with some regular instances. The nine devices are a (i) Danmini Doorbell,

(ii) Ecoobee Thermostat, (iii) Ennio Doorbell, (iv) Philips B120N10, (v) Provision PT-737E, (vi) Provision PT-838, (vii) Samsung SNH-1011-N, (viii) SimpleHome XCS-1002-WHT, and (ix) SimpleHome XCS-1003-WHT. Each device consists of sufficient records of attacks and regular instances with 115 features vector. This dataset is publicly available for download from [71] ML repository.

(iv) BoT-IoT

The BoT-IoT dataset [72] is created by simulating various legitimate and simulated IoT network traffic captured representing various attacks and normal behaviours. The attack traffic activities include Scanning (OS and service scan), Keylogging, Data exfiltration, DDoS, and DoS [73]. Features of this dataset are labelled based on the different categorization of attacks. The dataset contains 72 million records with 16.7 GB of CSV format file [72].

(v) Kitsune

Kitsune dataset consists of multiple traffic captured from nine commercially IP-based surveillance and an IoT network setting [74]. The dataset contains attacks that violate confidentiality, integrity and authenticity. These attacks are categorized into (i) Reconnaissance attacks, (ii) DoS attacks, (iii) Mirai attacks, and (iV) Man-in-the-middle attacks. In simulating the botnet attacks, nine Mirai Telnet scanner was used to infect IoT devices with the Mirai attack by exploiting their default credential and scanning new vulnerable victims' network. Kitsune dataset is made publicly available in the ML repository [75]. The dataset consists of 115 features that are categorized based on the captured attack traffic.

(vi) WUSTL

The WUSTL dataset consists of reconnaissance, command injection, DoS, Backdoor attacks and legitimate traffic that emulates real-world industrial IoT systems for cyber-physical systems security research [76]. This dataset is useful in investigating the feasibility of ML algorithms for detecting various real-world attacks. The raw data consists of 7,037,983 samples with seven (7) features. It comprised 93.30% for benign records with 6.7% attacks data records. A description of the statistics of attacks and normal traffic samples is available at [77].

(vii) IoT-DDoS

The IoT-DDoS [78] dataset is a DDoS botnet simulated dataset. It consists of simulated traffic flows from various IoT chatbots that infect IoT devices with DDoS attacks. The dataset contains various distributions of DDoS attack traffic to be used to investigate the feasibility of the ML algorithm. This can be an initial stage of exploring the capability of preventing DDoS botnet attacks in an IoT network environment.

Table 2.3 presents a summary of the utilised benchmark datasets with their various security attacks. Most of these datasets have captured the attack discussed in section 2.2.

Table 2.3: Benchmark Dataset

Literature	Name	Target Attacks	Technique(s)
[64]	KDD 99.	Dos and Scanning.	AIS, GI and PCA.
[67]	Unsw.	DoS, Reconnaissance and Backdoor.	LGBM.
[70]	NBa-IoT.	Scanning (Bashlite and Mirai).	AIS, LGBM, FCNN.
[72]	BoT-IoT.	DoS, Scanning, Keylogging and Data Exfiltration.	LGBM.
[74]	Kitsune.	Reconnaissance and DoS.	FCNN.
[76]	WUSTL.	DoS and Backdoor.	FCNN.
[78]	IoT DDoS.	DDoS.	FCNN.

2.3.10 Performance Metrics

As most of the available IoT benchmark datasets are often imbalanced, meaning the contains more traffic from one class than the other, many performance metrics are widely used for evaluations. Prominent of them includes the test accuracy, for evaluating an optimised model performance while benchmarking with the baseline model. However, other metrics are important to assess AI-based model performance in security applications. The F1 score corresponds to the harmonic mean of precision and recall is an important metric to justify model feasibility. It considers accuracy for each class sample. Employed metrics utilised the True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Accuracy, precision, recall and F1 score are defined in Equations 2.10, 2.11, 2.12 and 2.13.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.10}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.11}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.12}$$

$$F1score = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{2.13}$$

2.4 Detection Algorithms

AI techniques have been applied widely in the literature to address security challenges in the IoT [79]. Elrawy et al. [80] present recommendations for developing ML and DNN based IoT intrusion detection systems. These techniques remain promising in the field of IoT security monitoring research.

Bhunia and Gurusamy [81] utilise SVM to propose a framework for IoT security monitoring. The framework is determined to secure IoT devices at the network level by detecting anomalous behaviour within the network. A Mininet emulator was employed to set up the IoT networks and validate the proposed detection method. The proposed SVM method can detect network anomalies within the emulated IoT network traffic captured with a 98.00% precision value. Nskh et al. [82] employed dimensional reduction technique with an SVM classifier for network intrusion detection tasks. On empirical evaluation with the KDD 99 dataset, with five PCA components, they achieved a 98.50% accuracy value which is less than the accuracy without PCA utilization.

In the aspect of DTEM explorations, Lopez-Martin et al. [83] proposed an IoT network traffic forecasting technique based on the Stochastic Gradient Boosting (GB) classifier. The main idea is to use the ensemble models to improve the performance of various ML algorithms while determining the future state of network traffic flow as active or inactive. The proposed scheme can detect the active connection better than the inactive traffic flow. Moustafa et al. [84] integrated statistical ensemble-based approach aims to detect botnet attacks against the utilised protocols in IoT network traffic. They exploit IoT network traffic flow at the protocols level and subsequently apply ensemble ML-based to detect attacks within the captured traffics. They employed an Adaboost algorithm to assess the features' relevance in developing an appropriate model. Their proposal performs better for detecting botnet attacks with 99.54% as tested with the UNSW-NB15 dataset. However, it takes more testing execution times than conventional decision trees and Naive Bayes classification models. As such their method may not scale through IoT resource-constrained devices. Tang et al. [85] enhanced the Adaboost algorithm to detect low-rate DDoS attacks in an IoT environment. They considered LDoS attacks traffic captured dataset while evaluating their proposal. A Network Simulator Version 2 (NS2) was used for investigation purposes while assessing the model performance. Their method can detect attacks at a 97.06% detection rate. Resende and Drummond [86] outline RF employment opportunities in network security monitoring within the context of attacks detection and network data analysis. Hasan et al. [87] used it for network intrusion detection. Empirically validated the approach using an extended version of the KDD 99 dataset. Farnaaz and Jabbar [88] investigation considered an extended version of the KDD 99 dataset (NSL-KDD 99). The model can detect network traffic attacks accurately by 99.67%.

In an attempt to propose an effective security technique, Ikram and Cherukuri [89] employed a dimensional reduction technique with a SVM classifier for network intrusion detection tasks with a similar dataset employed by [88]. However, such implementations with older KDD 99 and its extended version may not provide a generic performance of the tested model. In addition, assessing the model using a single dataset may render its usability as ML depends on the datasets. Haripriya and Anju [90] explore AIS to identify attacks using a similar dataset. They utilised PCA to reduce features that were normalized based on min-max normalization [91, 92]. They reported the performance of AIS with large and smaller datasets. However, more details about which PCA features provide better performance compared to the usage of overall features are missing. In addition, they do not consider the computational cost of running their proposed method. We address these issues in our AIS with feature reduction empirical evaluation. In our work, we reported the memory and time consumption of using the overall data features and PCA reduced in training and testing scenarios. On the other hand, [93] utilised LGBM to detect DDoS attacks. Both LGBM and extreme gradient boosting (XGBoost) are empirically validated and performance comparisons were reported. Experimental results demonstrated that the LGBM predicts DDoS attacks without considering the UDP lag attack with 94.88% accuracy and consumes a lower processing execution time (30 seconds). However, sufficient details of the descriptions of the utilised LGBM parameters and hyperparameters and their selections and the experimental implementation are missing. At preprocessing stage, object columns such as Flow Packets, Flow Bytes, and Destination IP were removed from the utilised CIC-DDoS2019 dataset [94] to speed up the training procedure. In addition, their implementation considered only the CIC-DDoS2019 dataset [94]. These issues may restrict the feasibility of their proposal. Unlike their presented work, Okey et al. [95], integrate XGB and LGBM to detect cyber security attacks captured from different network traffics using similar dataset [94]. The proposed technique was compared with RF, DT, LGBM and XGB. They emphasized that their proposed method is lightweight. However, an investigation of the execution time to train and test their proposed model can assess the efficiency of their technique. As such, their proposal focused on performance evaluation rather than model efficiency. None of these implementations considered reducing the resources (memory and time) consumption of running LGBM algorithms for cyber security or IoT security monitoring purposes. A disadvantage of LGBM is that it can become overfitted before convergence. Such a procedure may restrict its capability to learn and interprets other important data features [96].

This motivates the explorations of DNN techniques to gain the advantage of its accurate classifications [97] while addressing the DNN model's resource-hungry limitations. Especially, the available DNN technologies in the literature used for IoT cyber security monitoring. Zhang et al. [98] DNN-based framework can detect the attacks captured from various Cyber-physical Systems (CPSs). They investigate such tasks for IoT data analysis. In the process, they frequently monitored the considered CPSs operations to trace any malicious activity. Fadlullah et al. [99] exploits state-of-the-art DNN approaches for IoT network traffic control and monitoring purposes. Chen et al. [100] used DNN to predict IoT network traffics data captured from the urban road network traffic dataset. Their scheme integrate Long Short Term Memory (LSTM) and Sparse Auto Encoder (SAE) to improve the prediction accuracy of the traffic flow captured. The SAE is utilised for dimensionality reduction, while the LSTM is for traffic identifications. Their method achieved 97.67% detection accuracy. Abdellah and Koucheryavy [101] utilises a similar DNN model, the LSTM to predict IoT network traffic captured from time series data. They empirically validated their model using Matlab implementation framework and reports their technique performance. However, no explanations concerning the tuning and chosen parameters of the implemented LSTM model, nor a report of its detection accuracy. Li et al. [102] explored DNN in the context of accurate classification and analysis of IoT smart cities data. They present concise data analytic capability of DNN along with open research directions. Their proposed approach can predict the captured data by 97.80%. Jung et al. [103] proposal determined to detect IoT botnets attacks. Their approach considered the sensory power consumption data, intending to analyse their behaviours and secured them against botnet attacks. They empirically evaluated their approach across selected IoT devices. Their method can accurately detect botnet attacks at the rate of 90.00%. Pour et al. [104] investigations aim to detect similar attacks from telescope device sensory data. Because of that, they utilised the Keras platform to evaluate their framework. As reported, their method can identify and classify the compromised traffics behaviour captured from the tested IoT devices with 90.85% precision value.

With regards to the device-level applications, Tang et al. [105] investigate the capability of using a compiler-based platform as an inference for bench-marking DNN on mobile devices. They exploited the concept and feasibility of implementing cloud-based DNN models. To this end, they utilised TensorFlow to run DNN on ARM-based platforms. In the aspect of resource management, Iandola and Keutzer [106] described the minimum speed requirements for deploying DNN in a resource-constrained environment. They outline various procedures for creating a small DNN architecture. They discuss the advantages of having such DNN architecture, especially in a resource-constrained environment. However, they focus on explorations rather than empirical evaluations. Shi et al. [107] utilised reinforcement-based DNN for resource management in an Industrial Internet of Things (IIoT) environment. Their method performs better than the benchmark counterparts in terms of limited spectrum utilization to meet the Quality of Service (QoS) requirements. Shen et al. [108] compressed CNN for structure learning in an IoT resource-constraint environment. The technique demonstrates its potentiality on the CIFAR-10 and Imagenet benchmark datasets. Lawrence and Zhang [109] proposed a similar compact architecture with low-precision floating-point computations in the convolutional layer. The lack of model assessments with IoT benchmark datasets and non-consideration of memory usage are the restrictions of their method's potentiality for deployment in a resource-constrained environment. Kodali et al. [110] utilised FCNN, for classification tasks on resource-limited devices. They evaluated their approaches on seven activity recognition datasets with 28nm mainstream CMOS technology. They utilised the performance metrics described in section 2.3.10 to assess their model performance against the utilised datasets. They described the utilised model size in the range of 0.104 MB for the smallest and 0.8000 MB for the highest that cannot scale through the constrained class illustrated in section 2.1. Moreover, the lack of consideration for model complexity while selecting the FCNN architecture may restrict method feasibility. In addition, their implementations that use non-IoT cyber security datasets focused on reducing energy consumption rather than computational resources (memory and execution) time. In addition to these related studies discussed, Table 2.4 presents a comparative summary of the existing ML and DNN methods used for IoT security monitoring and their limitations. These limitations include using one dataset to validate a proposed model, and not reducing the computational resources in training and testing a proposed model. In some scenarios such as [111], the memory and time usage was reported, however, it needs a further reduction to be deployed on ultra-low power devices.

Table 2.4: Comparison of ML and DNN techniques developed for IoT security

Literature	Technique Description	Dataset	Contribution	Limitation
[112]	Using AIS for benign samples selection	NSL-KDD [113]	Detection of attacks	old dataset No consideration of resource consumption
[114]	Optimizing LGBM parameters for efficient training	DS2OS [115]	LGBM provides better execution time (1.49 seconds)	Consideration of one dataset without considering the memory footprint
[111]	Optimizing LGBM to save computational resources	Self generated	Execution time and memory are 140.217 seconds and 347,530 bytes	Consideration of one dataset and higher execution time
[116]	Exploring DTEM to find better model	BoT-IoT [72]	LGBM requires 10 seconds in detecting attacks	LGBM not optimised and no consideration of memory usage
[117]	Reducing storage requirements for training RF and XGB	KDD 99 [66]	RF and XGB run within 5.7 seconds and 5.4 seconds	Non consideration of IoT dataset, non comparison with the baseline DTEM
[118]	utilised one class and multi-class SVM	Unsw [68]	Detection of IoT network attacks traffics captured	Consideration of one dataset and non consideration of resource usage
[119]	Integrating logistic regression model with FCNN	N-BaIoT [70]	The integrated logistic regression influence better attacks detection	Used few devices data of N-BaIoT and non consideration of resources consumption
[120]	Integrating PCA with CNN for IoT security	Unsw [68] and BoT-IoT [72]	Lightweight detection model with size 190 KB and 181 KB for Unsw and BoT-IoT, respectively	No consideration of training and testing computational resources
[121]	Pruning FCNN weights for efficient detection of attacks	KDD [66]	Detection of attacks in KDD	No consideration for the model computational resources
[122]	Enhancing FCNN for IoT attacks detection	DS2OS [115]	The model detects IoT attacks within 34.51 milliseconds	Used of one dataset, and non consideration of training and testing memory consumption
[123]	Integrating LSTM and CNN for IIOT security	Unsw [68]	Better performance of the integrated model	Using one dataset and non consideration of computational resources

With these limitations of ML models for IoT security monitoring, this thesis addresses these issues by exploiting LGBM to reduce its computational resources (memory and time) and utilised it for effective and efficient IoT attack detection. Then, it proposed an efficient LGBM training procedure that can save memory and time in training to create a lighter model with accurate IoT attack detection. In our implementation, we used four

IoT benchmarked datasets and compared our proposed optimised LGBM with its baseline counterparts and other conventional decision tree ML models. The resulting lightweight LGBM model offers a promise, as it saves testing memory and time resources better than each utilised technique. Furthermore, we address the resource consumption limitations of DNN with FCNN investigations by optimizing its training procedure to produce its resource-efficient counterparts with minimal memory and execution time. The optimised model can efficiently detect IoT attacks without accuracy degradation. In addition, most of these techniques compressed DNN by the quantization of weights and bias parameters. However, the proposed approach in this thesis targets effective attack detection with resource minimization to reduce FCNN (DNN-based) computational complexity. The method exploits pruning, simulated micro-batching and parameters optimization to regularize the resulting DNN model and reduce memory and time requirements while increasing accuracy performance. Unlike previous works that considered few dataset (see Table 2.4, in this thesis we considered multiple IoT device centric datasets with sufficient records of attacks and normal captured traffic. Therefore, the work presented in this thesis is novel and different from existing proposals in the literature.

2.5 Adversarial Attacks against AI

One of the limitations of using ML and DNN-based models in IoT security monitoring is that an attacker can impede the Confidentiality, Integrity, and Availability (CIA) of the deployed model using various attacks targeting the ML model itself. Despite the widest utilization of DNN for cyber security monitoring, DNN detection capabilities can be exploited easily by feeding the network with adversarial samples [124]. Looking at the architecture of the IoT network environment, we are more concerned to build a robust DNN-based security mechanism for IoT devices. This is useful for the proposal of robust, effective and efficient IoT security solutions that can be utilised to detect various cyber attacks. For this purpose, the perturbation methods used in crafting the adversarial samples are the Fast Gradient Sign Method (FGSM), and Projected Gradient Descent (PGD) [125], semantic [126], and random noise [127]. The FGSM attacking technique is computationally efficient. It required a one-step gradient update towards the direction of the gradient sign as in Equation 2.14. The notation X^o represents the original data, ϵ represents adjustment step of the original data, Y is the label, θ represents the model parameters, ∇_{X^o} is the backward propagation steps for gradient update, $J(l, X^o, Y))$ is the loss function used to train the network. The FGSM in Algorithm 2 can be used to generate a new perturbed dataset X^{fgsm} based on the original data X^o. The epsilon ϵ

Algorithm 2 FGSM perturbation procedure

```
    Input: $\mathcal{X}, \mathcal{Y}, m, \mathcal{M}$, data, label, epsilon length, model
    Output: Perturbed data $\mathcal{X}'$
1: for $\epsilon = 0$ to $m$; do    ▷ iterate over $\epsilon$ values, $m = 1$, for normalized data within $[0, 1]$
2:     $F_{\mathcal{M}}(\mathcal{X})$    ▷ Model $\mathcal{M}$ Forward propagation
3:     $B_{\mathcal{M}}(\mathcal{X}, \mathcal{Y})$ ▷ Model $\mathcal{M}$ Backward propagation for gradient update $(\nabla_{\mathcal{X}} J(\mathcal{X}, \mathcal{Y}))$
4:     $\mathcal{X}' = \mathcal{X} + sign(\epsilon * g)$
5:     $\mathcal{X}' = clip(\mathcal{X}', [0, 1])$    ▷ Clipping $\mathcal{X}'$ values within $[0, 1]$
6: end for
7: return $(\mathcal{X}')$
```

parameter determines the proportion of generated X^{fgsm} samples. These features are normalized using the clipping method to align with the corresponding X^o data features. The success of FGSM motivates the proposal of PGD [125]. PGD is an extended version of the FGSM method described in Equation 2.15. Both methods operate by computing the forward and backward propagation while generating perturbed samples. In the PGD attacking method, an initialized noise $\mathcal{U}(-\epsilon, \epsilon)$ based on uniform distribution of the ϵ is added to the original data sample before generating and clipping the adversarial samples repeatedly. Then, Equation 2.15 iterated t times to generate the perturbed samples. Where Π_{X^o+S} represents the projection of perturbation set $X^o + S$ using the projection operator Π, α is the gradient step size, J is the loss function.

$$X^{fgsm} = X^o + \epsilon * sign(\nabla_{X^o} J(\theta, X^o, Y)) \tag{2.14}$$

$$X^{pgd} = X^{t+1} = \Pi_{X^o+S}(X^t + \alpha * sign((\nabla_{X^o} J(\theta, X^o, Y))) \tag{2.15}$$

These adversarial perturbation methods can create craft samples to bypass the detection capabilities of various models. As such, they are appropriate techniques for evaluating the robustness of DNN models to examine their security monitoring capabilities in the IoT environment. For instance, an attacker can poison the training data before deployment by modifying a certain proportion of the label instance [128]. Such attacks consider feeding the model with poisonous training data. The intention is to reduce the classification performance [129], including the DNN model [130]. With that, a bunch of incorrect classifications are created by impacting the model used for security monitoring. In addition, a new set of perturbed data can be generated during the testing phase.

The Fast Gradient Sign Method (FGSM) proposed in [131] is a potential mechanism for launching such perturbation. Kurakin et al. [131] enhance FGSM and introduce Projected Gradient Descent (PGD). This is a targeted attack method that maximizes the probability of a specific target class. Hosseini et al. [126] proposed a semantic attack method to alter the meaning of the original data samples. With a given normalized data $X = x_i, i = 1, 2, ...n$ within [0,1], with n number of sample, by using the semantic attack, $X = 1 - x_i, i = 1, 2, ..n$. Athalye et al. [132] proposed a generic perturbation method based on randomly generated noise samples. With this attack, random noise is uniformly introduced into the data to compose the model. The noise is generated based on the distribution of the data. For normalized data within [0,1], the introduced noise will be in the form of $\mathcal{U}(0, 1)$. The Jacobian Saliency Map Attack (JSMA) based on the feature saliency map and Jacobian derivative computation of the learned DNN model can succeed in crafting adversarial samples [133].

These perturbation methods are white-box based, assuming the adversary has complete knowledge about the model applied in cyber security monitoring. Thus, they are widely applied in the context of IoT security monitoring [134]. Ibitoye et al. [135] utilised FGSM, and PGD to evaluate the robustness of Feed Forward Neural Network (FNN) and Self Normalizing Neural Network (SNN) by crafting the samples of the BoT-IoT dataset. Their evaluation shows that SNN can resist adversarial attacks better than its FNN counterparts. However, further explorations are needed. For instance, there was no consideration of the label-flipping attack technique and the analysis of the impacts of perturbation parameters ϵ on the success rate of the adversarially generated samples. A comparative study on the impact of adversarial attacks on network intrusion detection dataset is presented [136]. Their exploration investigates the performance of the ML-based model against the adversarial attacks generated using network security datasets. However, they only considered the conventional ML model and the adversarial attacks generated using IP-based datasets. We address these limitations by considering device centric dataset (N-BaIoT) in comparing the resilience of REDNN, FCNN and other conventional ML models against perturbed data samples. Unlike the investigations from [137] that considered testing the robustness of different DNN variants based on the adversarial attacks generated from the network security datasets, our explorations investigate the impact of different DNN architectures on resistant perturbed data samples. In addition, we investigate the influence of FP16 integration on the robustness of the DNN model to explore the feasibility of using a lightweight and robust DNN model in an IoT resource-constrained environment.

On the other hand, a robust and effective classification model can defeat various adversarial perturbations. Such a procedure considers training the built model with perturbed samples to augment regularization for resilience testing [138]. The approach used in this thesis tries to address these attacks for IoT security without utilizing the perturbed samples in training. The reason is to exploit the capability of the optimised FCNN model in defeating adversarial attacks effectively and efficiently. Therefore, we utilised eleven network traffic feature data from various IoT devices. These IoT benchmarks data capture relevant device-specific properties. Therefore, suitable for evaluating the performance of the proposed method with more realistic data samples instead of using oversimplified simulation methods. In chapter 5, we present details implementations of these various adversarial perturbations used in this thesis to test the resilience of our proposed method.

2.6 Federated Learning in IoT Environment

Federated Learning (FL) is a mechanism that leverages on-device model learning in a decentralized manner [139]. With FL, edge devices can learn and share a predictable model collaboratively without exposing their data to the cloud or fog for computations. This technique enables clients to retain their private data without exposing it to the data centre for specific operations while carrying out on-device training. Considering the IoT architecture discussed in section 2.1, FL can be a beneficial scheme for proposing AI-based security solutions in resource-constrained environments. This can be a strategy for developing feasible security solutions to detect attacks on IoT devices in real-time while granting privacy to each device's data. In FedAvg training, each client obtains its local Stochastic Gradient Descent (SGD), and a server performs the model averaging. At each communication round of training, a local gradient is computed based on randomly selected clients. The convergence of a model may depend on the number of communication rounds considered. As such, they can be an influential factor in producing a better global model. The function SERVER WEIGHTS UPDATE in Algorithm 3 describes the server aggregation procedure for K clients that return the averaged updated weights. In line 2 an initial weight value is defined before starting the local gradient training. Then, based on the number of predefined communication rounds in line 4, synchronised on-device training iterations with clients are performed to update their local weights in line 8. Depending on the sample size and clients associated sample data size, weights are averaged on line 11.

Algorithm 3 Federated averaging procedure

```
    Server Executes:
 1: function SERVER WEIGHTS UPDATE
 2:     initialize weight $w_0$
 3:     initialized $i = 1$
 4:     while $i \leq m$ do                              ▷ $m$ is the number of federated round
 5:         $n \leftarrow max(C.K, 1)$                    ▷ $C.K$ fraction of clients $K$
 6:         $R \leftarrow$ random set of $S_i$            ▷ $S_i \leftarrow$ random set of $n$ clients
 7:         for $k \in R$ in parallel do                 ▷ $k$ client index, a selected clients from $R$
 8:             Weight update for each client $k$
 9:         end for
10:         Averaged weights update ▷ Average weights update based on client $K$ weights
11:         $w_{i+1} \leftarrow \sum_{k=1}^{K} \frac{m_k}{m} w_{i+1}^{k}$    ▷ $m_K$ = client $k$ sample size, $m$ total sample size
12:         $i = i + 1$
13:     end while
14:     return Averaged updated weights
15: end function
```

The success of AI-based techniques, especially the DNN recently, motivates researchers from several disciplines to explore FL paradigms from different perspectives. In the field of IoT security monitoring, FL is gaining popularity. Preuveneers et al. [140] explored FL applications for intrusion detection in IoT networks. Lim et al. [141] and Imteaj et al. [142] describe open research problems on FL for resource-constrained IoT devices. Nguyen et al. [143] proposed a signature-based FL method to detect attacks on IoT devices. In that context, benign activities are profiled to create the desired intrusion detection framework. A deviation from the benign behaviour of the actual device indicates malicious activity for that particular device. Liu et al. [144] exploit FL capability to detects attacks on Industrial IoT (IIoT) devices. They utilised a labelled dataset for training a DNN model in a federated manner to identify malicious attacks. In addition, they integrate CNN and Long Short Term Memory (CNN-LSTM) for better model convergence. However, the MNIST and CIFAR-10 datasets utilised for estimating the model parameter gradients are non-IoT data. Jiang et al. [145] utilised model pruning for efficient FL training on edge devices. Similar to Liu et al. [144] work, they considered image dataset. Bonawitz et al. [146] proposed a TensorFlow-based FL framework for mobile devices. For evaluation, they utilised Android mobile devices. Popoola et al. [147] used FL to detect a zero-day attack in an IoT network environment. Their implementation takes the advantage of FL data privacy without considering resource limitations. In their investigations, they considered the N-BaIoT [70] device-centric dataset. However, none of these proposals considers optimizing FL training to reduce memory consumption on IoT networks using pruning, micro-batching, and parameter regularization. This thesis address this challenge by optimizing the federated training procedure using raw network traffic datasets from various IoT devices captured on N-BaIoT [70] dataset. Then, it proposed a REDNN FL method with minimal resource consumption. This method maintains state-of-the-art accuracy while reducing memory consumption. An edge decentralized network testbed was developed to address the deployment issue. Regarding the generalizability, non-IoT datasets are considered.

2.7 Chapter Summary

After an introduction to the IoT environment in the context of applications, architecture communications behaviour and security issues in a resource-constrained environment, this chapter explores various AI techniques. It starts with the ML which is a broader form of AI followed by the DNN which is the subset of ML. In each case, the discussion of various AI-based detection techniques utilised for IoT security in this thesis is presented. This includes recent state-of-the-art ML and DNN used for IoT cyber security with their limitations. From the literature, the existing IoT cyber security ML methods used to identify malicious activities are resource-hungry. Previous studies consider addressing these issues while employing one dataset or reducing the execution time usage, especially in model training or testing. However, they are not reducing the memory and time usage in both the training and testing phases. In addition, most of the datasets utilised for evaluation are general-purpose network security data, not from designated IoT devices. None of the existing works built a generic training procedure to reduce the resources (memory and time) consumption of running ML algorithms for cyber security or IoT security monitoring purposes using multiple benchmark datasets from commercial IoT devices. Even those that reduced the model computational expensiveness, further assessment can be useful for a feasible deployment on IoT resource-constrained environments.

The limitation of recent DNN work on IoT resource-constrained environments is the consideration of quantization or model pruning in reducing model complexity. As discussed in the literature, some existing work focuses on reducing the model size without considering training and testing resource consumption. In addition, most of the previous robustness evaluations of using DNN in IoT have not investigated the impacts of architectures and low precision (fp16) integration on robustness. As such, none of these discussed DNN works built an optimised training method using pruning, simulated micro-batching and parameters optimization to train the FCNN model in a robust, effective and efficient scenario. Such implementation is presented in this thesis using commercially tailored IoT traffic datasets, suitable for on-device model evaluation. The success of the proposed optimised DNN training procedure motivates further exploration of the on-device FL concept to benefit resource minimization while addressing security, data privacy and confidentiality in an IoT environment. This is to overcome the commonly used simulation method of testing the deployment capability of the FL model in an IoT environment.

Chapter 3

Lightweight ML Method for IoT Cyber Security

Following the discussion and explorations of conventional ML algorithms in chapter 2 in the context of IoT security monitoring, it is clear that existing ML works have limitations. These limitations include computational resource consumption in training ML methods and effective detection with fewer resources (see Table 2.4). Therefore, this chapter proposed a lightweight ML algorithm for IoT security monitoring to overcome the limitations of existing works. It starts with careful investigations of the feasibility of feature reduction methods for IoT cyber security monitoring. It later explores the potentiality of proposing an optimization method for training ML algorithms to create an efficient and effective IoT security monitoring method. This can be an avenue for further explorations on the capabilities of improving ML to assess their feasibility, potentiality and performance as AI security-based solutions that can be deployed in IoT resource-constrained environments.

3.1 Resource Reduction Method

Reducing computational complexity in the process of training resource-hungry ML algorithms is essential for developing lightweight models. In particular for building effective and efficient classification models. For instance, the ML algorithm utilized for IoT security monitoring supposes to detect attacks efficiently using minimal resources. Because of that, attentive feature investigation of the IoT network traffic data is a requirement to reduce ML computational expensiveness. In this case, exploiting IoT feature extraction mechanisms can enable efficient network traffic classification for IoT cyber security monitoring. For this purpose, we integrate PCA with the (RNSA)-AIS-based algorithm aiming to reduce resource consumption in processing IoT data. The argument raised is whether the computational cost of applying PCA data brings any advantage compared with processing the original data. In addition, we integrate GI with the RNSA algorithm as an approach to extract highly relevant features in the Danmini Doorbell (DD) device data of N-BaIoT [70] and KDD-99 data [64]. Even though the KDD-99 dataset is an old cyber security dataset, it contains the most dangerous DoS attacks, and previous researchers used it to evaluate AIS model [90, 112]. Therefore, we utilized it to investigate and compare the resource consumption of the feature reduction methods applied with the AIS algorithm. This served as a procedure to assess the performance of feature reduction methods used to reduce the computational complexity of resource-hungry ML algorithms on multiple datasets and provide further investigations than [90] existing work that neglects the computational cost of running the AIS algorithm. Unlike [90] work, we further considered the feasibility of integrating GI in savings computational resources of training and testing AIS.

Lightweight RNSA Model Design

For building the lightweight RNSA model, we utilized the hypersphere space. This is to integrate the resource reduction approach into the RNSA algorithm. At training, the dimensions of employed datasets and each hypersphere present remain the same. As a result, each hypersphere parameter represents a data instance in the search space. The parameters are defined using a real-valued that represents a class label. The parameter radius value of the hypersphere is within the ranges of [0,1]. These values can be used to generate random samples using a distribution of benign instances. In our case, we tested various values from [0.1, 1] incremented by 0.1 to cover the entire search space. Then, 0.3 was returned as the best choice for the parameter radius. We used this returned value to generate uniformly distributed random samples between [0,1] based on benign class. The euclidean distance in Equation 3.1, with x_1 and x_2 data points, can calculate the distance between the randomly generated and real benign data samples in a search space. This is useful to determine the actual class of a particular data feature represented in the search space. For the PCA and GI implementation, we used the scikit-learn python module documentations [148]. The PCA and GI are fitted into the N-BaIoT and KDD-99 datasets to reduce feature data.

$$edist(x_1, x_2) = \sqrt{(x_1 - x_2)^2} \tag{3.1}$$

3.1.1 Experimental Procedure

We used Python 3.76 on a desktop computer with Intel Xeon E5-2695(4 core) CPUs running at 2.10 GHz with 16.0 GB for experimental settings. A memory profiler was used to record the model's memory usage [149]. Experimental records are investigated based on the deterministic properties of the data features in terms of resource minimization, the overall amount of N-BaIoT data, and the reductions achieved. For further explorations and reproduction purposes, codes for these experiments are publicly accessible [150].

Datasets and Preprocessing

The device-centric dataset DD examined is a subset of the N-BaIoT dataset [70]. The utilized dataset contains 1,018,298 data samples and 116 features, including the class label - attacks as '0's and the normal as '1'. The records are for both benign and malicious traffic, and each record represents a numeric traffic flow from a real network. For the task of cyber security monitoring, the KDD-99 dataset [64] was tested and examined.

The KDD-99 dataset contains some categorical features. The categorical features are transformed into numeric using LabelEncoder technique [151]. With n number of target classes, this technique can transform categorical features into numeric values between $[0, n-1]$. After checking for missing and identical feature values, each utilized dataset is separated into 80% for training and 20% for testing samples. These benchmark datasets investigated for the task of binary classification and resource reductions contains numeric traffic flow. As such scaling the features can preserve the relationship among the data samples as pre-procsing procedure. Thus, data features are normalized within the range of [0,1] using *Min-Max* normalization formula presented in Equation 3.2. With n data features $x_1, x_2, ..., x_n$, within a dataset, normalization is performed using the formula in Equation 3.2. The notation x_i', represents the normalized value of the i^{th} feature, x_i the original value, while min_{x_i} and max_{x_i} represents the minimum and maximum value of the i^{th} feature over the entire dataset.

$$x_i' = \frac{x_i - min_{x_i}}{max_{x_i} - min_{x_i}} \tag{3.2}$$

3.1.2 Experimental Results (Resource Reduction)

Figure 3.1 and Figure 3.2 illustrate the PCA variance ratio of the KDD-99 and N-BaIoT datasets, respectively. The PCA transformation has indicated that using the extracted number from 10 to 20 principal components from the N-BaIoT data, about 99% of the variance ratio can be retained. This indicates that fewer traffic features from the data can be sufficient for ML model training. We are more concerned about the principal components of the data that capture most of the variance ratio to use them and investigate the resource-saving and attack detection capability of the RNSA algorithm. In addition, Figure 3.3 and Figure 3.4 illustrate the GI features of the KDD-99 and N-BaIoT datasets, respectively. It is apparent that using only 26 GI features from the entire N-BaIoT data can be sufficient to build our ML model.

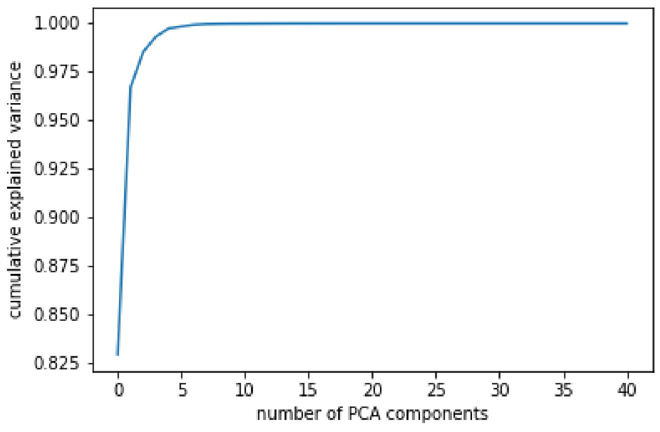

Figure 3.1: PCA components for KDD-99 data

Table 3.1 illustrate the accuracy values of the attack detection with and without feature reduction across each dataset. Interestingly, the feature reduction techniques used on the N-BaIoT and KDD-99 datasets do not decrease the detection accuracy value without the percentage points compared with the entire dataset. It increases the detection accuracy slightly. This validates the illustration in Figure 3.2 and Figure 3.4 while supporting the argument of reducing features in a dataset can reduce resource consumption with better or without significant performance degradation. This is because in our implementation, we consider the scenario which retained the 99.9% of the variance ratio during the feature selection.

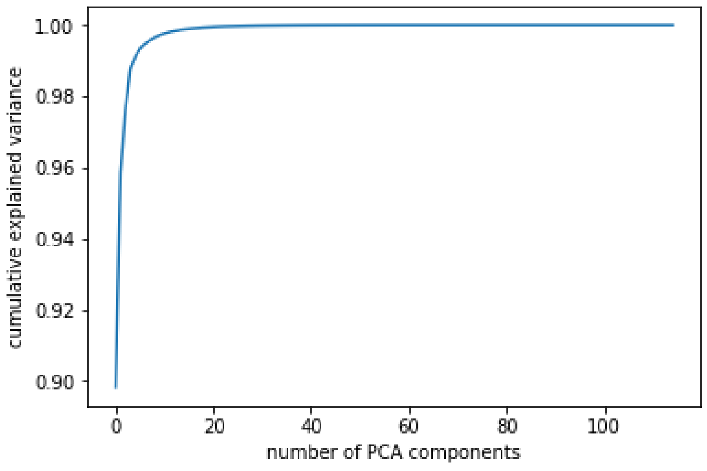

Figure 3.2: PCA components for N-BaIoT data.

Table 3.1: Experiments and datasets.

Datasets	Features	PCA	Gini	Accuracy (%)
N-BaIoT	115	N/A	N/A	68.30
	20	✓	N/A	68.80
	15	✓	N/A	68.50
	10	✓	N/A	68.60
	26	N/A	✓	68.50
KDD-99	41	N/A	N/A	80.00
	11	N/A	✓	80.10
	5	✓	N/A	80.25

Table 3.2 provides results of the samples training and testing memory consumption reported in the unit of Byte (B), with and without using feature reduction against the N-BaIoT dataset. In each case, the training and testing memory consumption for the RNSA demonstrates significant reduction using the integrated feature reduction method. In particular, with 10 PCA, more than 80% of the memory can be saved during the training and testing phase. This is because of using the lowest amount of features that capture all the variance in the dataset, in our case 99.9%. Looking at the memory requirement of constrained IoT devices illustrated in chapter 2, the feature reduction results demonstrate that AIS integrated with PCA can be utilized in IoT resource-constrained environments. For instance, it required 0.28626 Kilo Byte (KB) and 1.146 KB to train and test AIS with 10 reduced PCA feature records of the N-BaIoT dataset. The memory consumption results demonstrate the capability of running and deploying AIS with reduced PCA

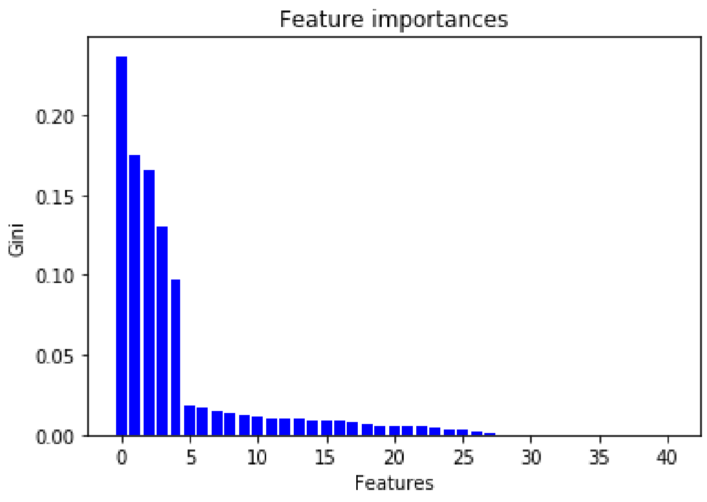

Figure 3.3: Gini Index feature importance for KDD-99 data.

features on most constrained devices in Class 0 (see Figure 2.3). The more testing resource consumption is due to the nature of the AIS algorithm in classifying IoT traffics based on the feature of the benign class that are generated in training. For more details about the profiling of the memory usage in training and testing (see [150] repository) with dedicated AIS prediction phase.

Table 3.2: Computational (per record) memory comparisons against the N-BaIoT dataset.

Computation	Full Features	PCA 10	PCA 15	PCA 20	GI 26
Training Memory (B)	1435.5	286.26	350.83	389.13	483.53
Testing Memory (B)	5758.1	1146.0	1405.3	1558.9	1945.9
Training Saved (%)	N/A	80.08	75.56	72.89	66.32
Testing Saved (%)	N/A	80.10	75.59	72.93	66.21

The runtime in milliseconds (ms) of the RNSA algorithm, with and without feature reduction against the N-BaIoT is presented in Table 3.3. The reported values are the training and testing execution times consumed with each technique. Considering the 10, 15, and 20 PCA components and the 26 GI features, the runtime is lowest in the case of using 10 PCA components compared with the remaining feature reduction approaches, with a total saving of 52.94% and 48.35% for the training and testing phases, respectively. As expected, the analyzed results reveal considerable reductions in memory consumption and processing time when using reduced data features. These results demonstrate the capability of the proposed approach in utilizing and managing ML resource consumption.

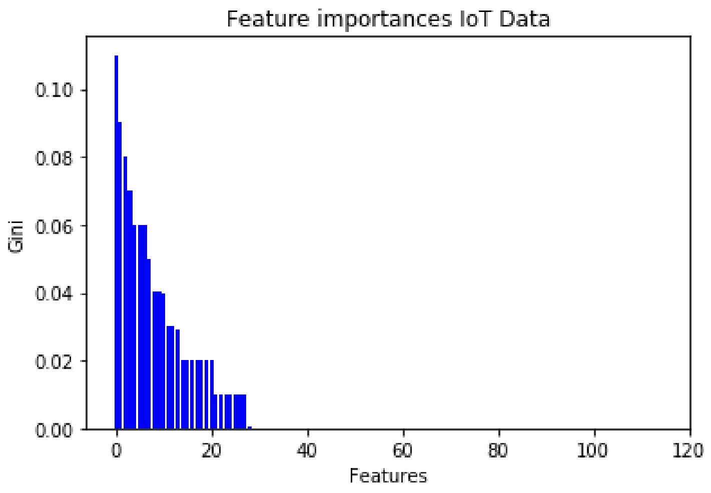

Figure 3.4: Gini Index feature importance for N-BaIoT data.

Table 3.3: Computational (per record) training time comparisons against the N-BaIoT dataset.

Computation	Full Features	PCA 10	PCA 15	PCA 20	GI 26
Training Time (ms)	300.52	141.41	154.69	172.35	223.92
Testing Time (ms)	547.97	283.00	300.56	336.00	427.36
Training Saved in %	N/A	52.94	48.53	42.65	25.49
Testing Saved in %	N/A	48.35	45.15	38.68	22.01

This can be a beneficial step in assessing the computational expensiveness of running ML algorithms for cyber security monitoring. The result attempt to answer RQ2 in terms of training ML method to be resource-efficient, however, the detection accuracy reported against the N-BaIoT dataset in Table 3.1 may limit the utilization of feature reduction techniques on IoT datasets for the proposal of efficient and effective IoT security monitoring methods.

3.2 ML Training Optimization

Following the limitations of existing works for IoT security monitoring and the encouraging results of the previous section, it is clear that an optimized method of training ML algorithms for effective and efficient attack detection is required. The feature reduction method may have limitations in creating lightweight ML algorithms, as some of the IoT data may not contain irrelevant and redundant samples and noise [152]. Redundant instances include repeated data samples, while irrelevant data are less useful features with no contribution to the learning phase of the ML model. In some cases, it is not even possible to know which traffic features are more relevant. In addition, resource utilization through feature reduction may not increase the detection accuracy of some algorithms, depending on the nature of ML models, as some require large data samples to perform better. These issues may limit the feasibility of using feature reduction for effective IoT security monitoring. To this end, we built an optimized ML boosting training procedure based on supervised learning. We adopted a more promising ML (LGBM) algorithm [153] from the DTEM method and obtained its less computationally expensive training parameters to build an effective IoT security monitoring-based solution demonstrated in section 3.2.1 and 3.2.2. For proof of concept, we used LGBM with the optimized training procedure to create its efficient and effective counterparts that can accurately detect IoT attacks.

3.2.1 LGBM for IoT Security Monitoring

The procedure to optimize the DTEM classification model can be considered a challenging task. This is due to the need for intensive parameter tuning in achieving state-of-the-art performance. For example, optimization of the gradient boosting method requires attentive parameter tuning to build a base learner that is maximally correlated with the negative gradient of a loss function [154]. In this aspect, careful investigation of LGBM DTEM-based parameters described in section 2.3 is a requirement to build an effective model. Regarding efficient resource utilization, the learning rate and the regularization term are selected in the range of [0.0001, 0.1]. In addition, the constraint bagging and feature fraction are utilized within [0, 1]. The value of 2 assigned to the constraint number of leaves can avoid model over-fitting while reducing the computational cost of training the LGBM model. Proper configuration of these chosen hyperparameters can minimize time and memory resource consumption. Traditionally, the grid search technique can select the best parameter settings among various learning models. However, the grid search aims to improve the prediction performance rather than the trade-off between memory usage and accurate prediction. Because of that, an optimized LGBM training procedure can be used to propose an approach for effective IoT security monitoring that reduces memory and time resource consumption. The visualization in Figure 3.5 is the process diagram of such a method (see Algorithm 4). It required a dataset and the baseline LGBM model alongside their trainable parameters. This training procedure can

Algorithm 4 Resource efficient LGBM

```
    Input: Labelled dataset $\mathcal{D}$, Model $\mathcal{M}$, Set of parameters and hyperparameters $\mathcal{P}$
    Output: Efficient model $\mathcal{M}_e$
 1: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2),..., (x_n, y_n)\}$
 2: Normalized $\mathcal{D}_n = \{D_j\}, j = 1 \dots n$
 3: Train data $X \subseteq \mathcal{D}_n$
 4: Test data $X' \subseteq D_n$                                ▷ $X \cup X' = \mathcal{D}_n, X \cap X' = \emptyset$
 5: $\mathcal{P} = \{p_j\}, j = 1 \dots l$                 ▷ $\mathcal{P}$ model parameters for $\mathcal{M}$, $l$ parameter length
 6: $m_f$, $t_f$ fitted memory and fitted run times of $\mathcal{M}$
 7: function EFFICIENT($\mathcal{P}$)
 8:     $t_{fp_1} \leftarrow t_{C(p_1,T)}$                    ▷ Run time to fit $\mathcal{M}$ with initialized $p_1$, and $X$
 9:     $m_{fp_1} \leftarrow m_{C(p_1,T)}$               ▷ Memory footprint to fit $\mathcal{M}$ with initialized $p_1$, and $X$
10:     $min(t) \leftarrow t_{fp_1}$
11:     $min(m) \leftarrow m_{fp_1}$
12:     for $k \leftarrow 2, l$ do                         ▷ Iteration using $\mathcal{P}$ parameters value
13:         $t_{fp_k} \leftarrow t_{\mathcal{M}(p_k,T)}$
14:         $m_{fp_k} \leftarrow m_{\mathcal{M}(p_k,T)}$
15:         while $(m_{fp_k} \leq min(m)))$ do
16:             $min(m) \leftarrow m_{fp_k}$
17:             if $((t_{fp_k} \leq min(t))$ then
18:                 $min(t) \leftarrow t_{fp_k}$
19:             end if
20:             $\mathcal{M}_e$ = trained model that estimate ($min(t)$ and $min(m)$)
21:         end while
22:     end for
23:     return ($\mathcal{M}_e$, min(m), min(t))
24: end function
```

accept data, normalize it, and output an efficient LGBM model M_e with minimal memroy footprint and execution time. Each utilized dataset is produced from the collection of realistic benign and malicious traffics using simulated IoT devices. At training, the optimized procedure in Algorithm 4 can select the optimum parameters of the LGBM model defined in line 5 based on minimal fitted memory and running times constraint defined in line 6. At implementation, combination of these parameter are stored in a dictionary, and selection are based on each training iteration. The selection procedure described in Algorithm 4, utilizes the function efficient from lines 7 - 24 which takes trainable parameters from the parameters dictionary as arguments to initialize the memory and time consumption of the baseline classifier model $\mathcal{M}$ based on fitting the model with the first parameters combination p_1 from $\mathcal{P}$. Then, iterate sequentially to the remaining set of parameters in $\mathcal{P}$ from lines 12 based on the length of the available parameters store in a dictionary to return an efficient LGBM model. While line 15 remains true, in line 16, we update the estimated model memory footprint with the minimal value obtained by the baseline classifier. The estimated execution time is compared with that of the baseline in 17 and update the minimal requirements in line 18. This is to find the best model parameter combinations with minimal training resource consumption. Depending on the utilized dataset, this model is employed to predict a testing dataset with better or state-of-the-art accuracy as described in Algorithm 5. The description of the testing procedure is in Algorithm 4. It uses the efficient model $\mathcal{M}_e$ returned by Algorithm 5 to predict a given testing data. The purpose of Algorithms 5 and 4, is to reduce the computational cost of running LGBM for IoT security monitoring. In particular, to automatically iterate through a set of parameter dictionaries and select the most efficient one with minimal memory and time consumption. The emprical evaluation of these algorithms considered IoT datasets, and redcuced the computational memory footprint to fit the requirements of IoT devices with better accuracy performance (see section 3.2.3). The return parameters chosen by our training procedure are in Table 3.5. This can be an initial step towards algorithm deployment in resource-constrained enivironment.

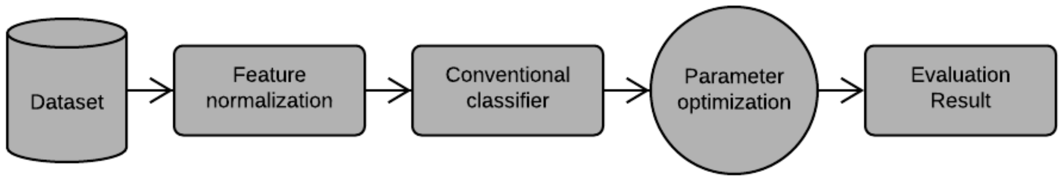

Figure 3.5: Resource efficient LGBM process diagram

Algorithm 5 Proposed algorithm to obtain effective and efficient LGBM

```
    Input:  Model $\mathcal{M}$, $\mathcal{D}_n$, $\mathcal{P}$
    Output: Efficient validation model $\mathcal{M}_v$
1: Test data $X' \subseteq D_n$                          ▷ $X \cup X' = \mathcal{D}_n$, $X \cap X' = \emptyset$
2: for $i = 1$ to $m$ do                                 ▷ $m$ is the length of $X'$
3:     EFFICIENT($\mathcal{P}$)                          ▷ Call to EFFICIENT in Alg. 4
4:     $\mathcal{M}_v = \mathcal{M}_e$                   ▷ $\mathcal{M}_e$ returned by function EFFICIENT in Alg. 4
5:     Prediction $E = \mathcal{M}_v(X')$                ▷ Prediction using $M_v$ and $X'$
6:     estimate prediction memory, prediction time       ▷ based on efficient model $\mathcal{M}_v$
7: end for
8: return ($\mathcal{M}_V$)
```

3.2.2 Evaluation

In this section, we present the experimental evaluation of the proposed method with the description of the benchmark datasets selected for IoT security monitoring. We started with details description of each utilized datasets follows by preprocessing technique utilized on each dataset.

Utilized Datasets and Preprocessing

The evaluation experiments used four accessible IoT datasets. These are the N-BaIoT [70], the Bot-IoT [72], the Bot-10 [72], and the Unsw [67]. Each dataset consists of various attacks along with normal traffic activities. Particularly, Bot-IoT with multiple categorizations of different botnet attacks, 10% of the data is extracted from [72] to create the BoT-10 dataset for model evaluation. The choice of these datasets allowed frequent model training for a thorough investigation. Each tested dataset is categorized into 70% training and 30% testing records. The datasets are described briefly in chapter 2, (see Table 2.3) while their training, testing and feature dimensions are illustrated in Table 3.4. All data records are normalized using the employed min-max standard normalization formula described in Equation 3.2.

Implementation

In practice, the utilization of the parameter grid [155] is due to the various series of training sessions that occur for the discovery of optimum parameters with fewer resources. The parameter grid [155] can store multiple parameters. Training and testing were implemented on Spyder [156] version 3 stable python IDE.Memory footprints and iteration time are profiled using the integrated *psutil (process and system utilities)* and *time* python modules. Computation was conducted on a personal desktop computer with the specifications described in section 3.1.1. For reproduction purposes, the experimental codes for the approach implementation are made publicly accessible [157].

Table 3.4: Utilized IoT datasets.

Dataset	Training	Tests	Feature dimensions
N-BaIoT	509,865	218,514	115
Unsw	115,264	49,400	43
Bot-IoT	467,965	200,557	34
Bot-10	1,247,596	534,684	10

Optimized LGBM Hyperparameters

The most relevant hyperparameters discovered using the proposed training method are described in Table 3.5. These include the initial values for the optimized LGBM model and optimum values returned by the efficient boosted algorithm.

Table 3.5: Initial and optimum hyperparameters.

Hyperparameter	Initial	Optimum
Feature Fraction	0.1	0.4
Bagging Fraction	0.1	0.4
Bagging Frequency	2	2
Number of Leaves	31	2
Learning Rate	0.1	0.0001
Regularizer	0.0	0.0001

3.2.3 Results

This section presents the analysis and discussion of the experimental results of the proposed LGBM method. It provides the memory and time usage comparison between the optimization procedure used to train LGBM against the baseline method and each tested technique.

Testing Time

The visualization in Figure 3.6 is the testing times needed to evaluate the proposed optimized LGBM method against each dataset record. It is efficiently faster than the baseline model. It demonstrates reduced classification time in processing each sample of the tested data. As a result, the proposed method is capable of saving processing time across datasets. As compared with the conventional model in Figure 3.6, it saved 53.79%, 57.89%, 61.11% and, 47.76% of times for validating a sample of Bot-10, Bot-IoT, Unsw and, N-BaIoT, respectively. As expected the optimized LGBM method that utilized our training procedure in Algorithm 4 and 5 demonstrate efficient performance and better resource minimization.

Testing Memory

The memory unit consumption comparison in testing each data record using the proposed approach is presented in Figure 3.7. The technique demonstrates minimal memory usage. It saved 52.69%, 39.17%, 71.43% and 41.78% of test memory for each record of Bot-10, Bot-IoT, Unsw, and N-BaIoT, datasets, respectively. These results indicate its robustness

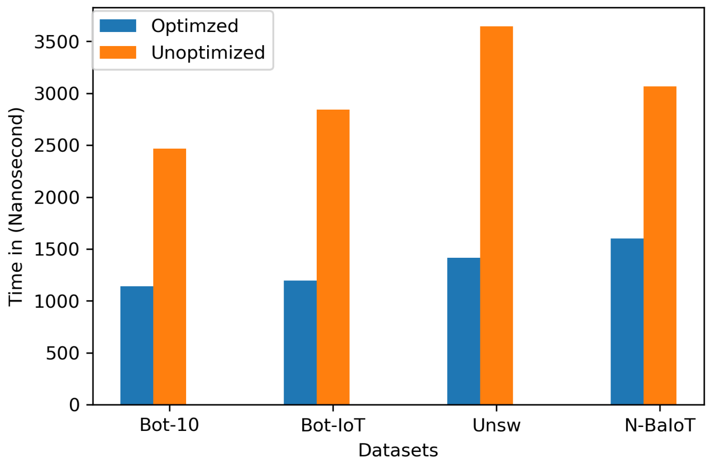

Figure 3.6: Sample testing time resource consumption of LGBM: Optimized vs Unoptimized.

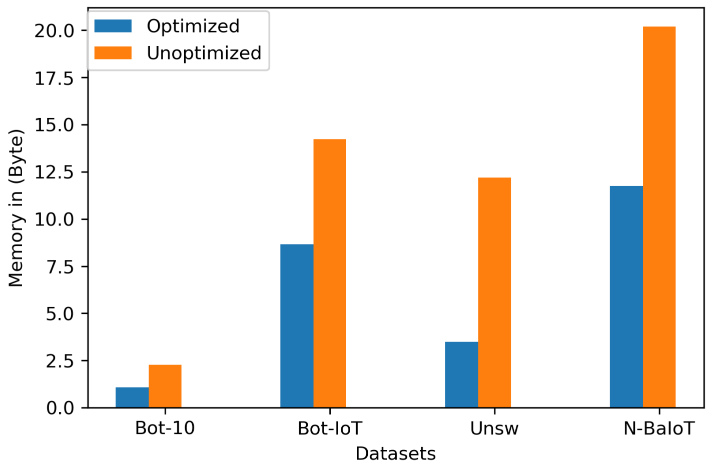

Figure 3.7: Sample testing memory resource consumption of LGBM: Optimized vs Unoptimized.

and lightweight security monitoring advantages for IoT devices with overall improvement across benchmark datasets. It suggests that IoT security monitoring with the proposed approach can be beneficial.

Testing Accuracy

The illustration in Figure 3.8 represents the testing accuracy that the proposed method

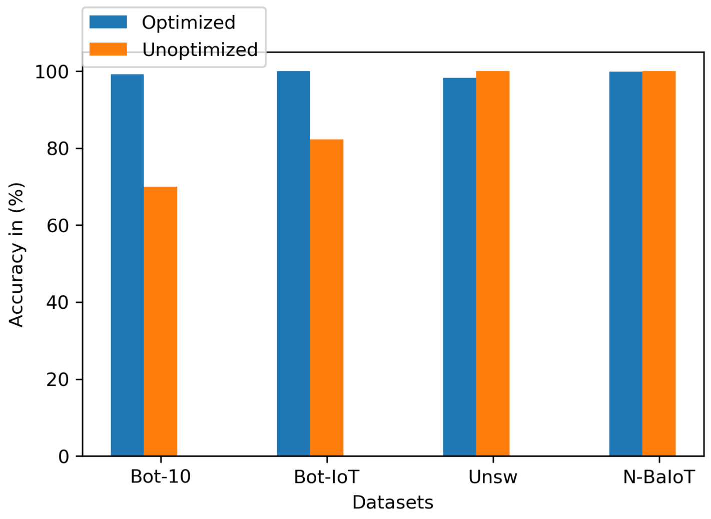

Figure 3.8: LGBM testing accuracy comparison: Optimized vs Unoptimized.

provides for each dataset. Despite its resource reduction advantages, it also outperformed the unoptimized LGBM method in predicting the Bot-10 and Bot-IoT datasets accurately. We notice a slighter accuracy degradation using the Unsw dataset. The reason is due to the while loop in Algorithm 4 that iterates based on minimal memory. The loss of accuracy by the unoptimized approach using the Bot-10 and Bot-IoT datasets was due to the utilization of the default configuration training parameters. These results indicate the capability of the optimized technique in discerning IoT attack traffic effectively. The illustration in Figure 3.9 represents the testing accuracy comparison of the proposed method and the grid search technique. Despite its less computationally expensive, the prediction accuracy across each dataset is closer to that of the grid method.

Computational Performance Analysis

The reports in Table 3.6 is the performance comparison of the grid search and the proposed optimized LGBM method against the tested datasets. Regarding the resource consumption of each data record, the proposed optimized LGBM approach is better. It required minimal memory and maintained reduced classification time reported in Nanoseconds (ns). These results indicate its effectiveness and efficiency advantage. It demonstrates the importance of utilizing our optimized training procedure over the conventional parameters optimization technique (the grid search).

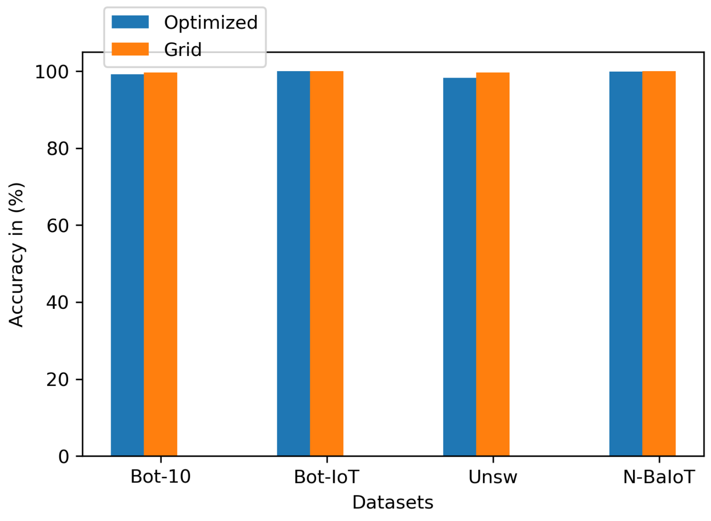

Figure 3.9: Optimized testing accuracy comparison with grid search.

Table 3.6: Grid and optimized LGBM methods performance evaluation (per record).

Dataset	Algorithm	Test memory (B)	Test time (ns)
N-BaIoT	Grid	15.88	2059.36
	Optmized LGBM	11.75	1601.73
Unsw	Grid	5.72	1619.43
	Optmized LGBM	3.482	1417.00
Bot-IoT	Grid	11.19	1495.83
	Optmized LGBM	8.66	1196.67
Bot-10	Grid	1.52	1458.81
	Optmized LGBM	1.08	1140.86

The description in Table 3.7 is the computational performance comparisons of the employed algorithms using the N-BaIoT dataset sample. The proposed optimized LGBM method indicates better memory and time resource savings compared with each tested technique. As the results are expressed based on how each model processes its data record, optimized LGBM provides the lowest testing memory footprints. In addition, it demonstrates a better classification of attacks and regular traffic with an accuracy of 99.90%. This suggests that optimizing ML model parameters in training can produce a better predictive model. The reason may be due to the efficient function procedure used in line 3 of Algorithm 5. In each iteration, the accuracy is computed using selected parameters. This is useful in selecting a better model with low computational resources.

Table 3.7: Models performance evaluation comparison on N-BaIoT Dataset (per record).

Algorithm	Accuracy (%)	Test time (ns)	Test memory (B)
Random Forest	89.35	12813.82	3873.01
LogitBoost	89.14	13042.64	3874.00
SGB	89.53	13454.52	3875.00
AdaBoost	89.31	11440.91	3866.00
Optmized LGBM	99.90	1601.73	11.75

The results demonstrate that optimizing the LGBM model to save its computational resource may not reduce its accuracy performance. Therefore, it validates the argument raised in RQ2 of training ML method to be resource-efficient for IoT security monitoring.

We investigate the relationship between hyperparameters tuning against memory consumption for each data record of the N-BaIoT dataset. Figure 3.10 indicates the effects of varying the constraint number of leaves on memory usage. As expected, the graph suggests that smaller values for tree leaves can influence better memory savings. Also, Figure 3.11 indicates the memory consumption while altering the bagging frequency parameter. An appropriate choice of bagging frequency combined with other parameters can be considered during training, particularly during model tuning for better resource savings. However, value selections depend on the dataset utilized and other tested parameters. The visualization in Figure 3.12 is the memory consumption against the constraint bagging fraction using the N-BaIoT dataset.

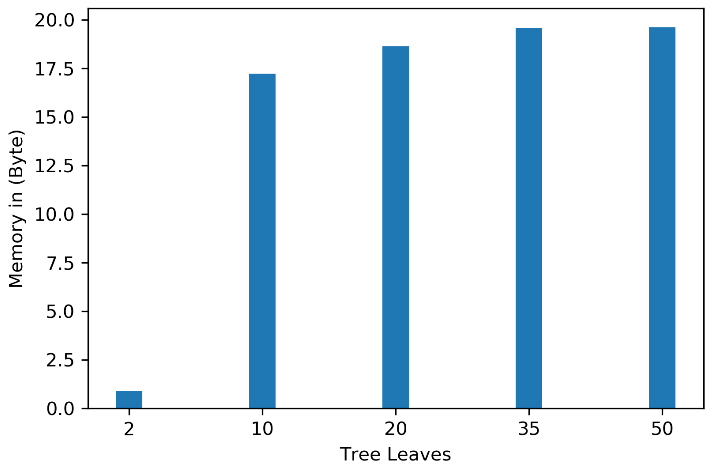

Figure 3.10: Effect of number of tree leaves on memory with N-BaIoT.

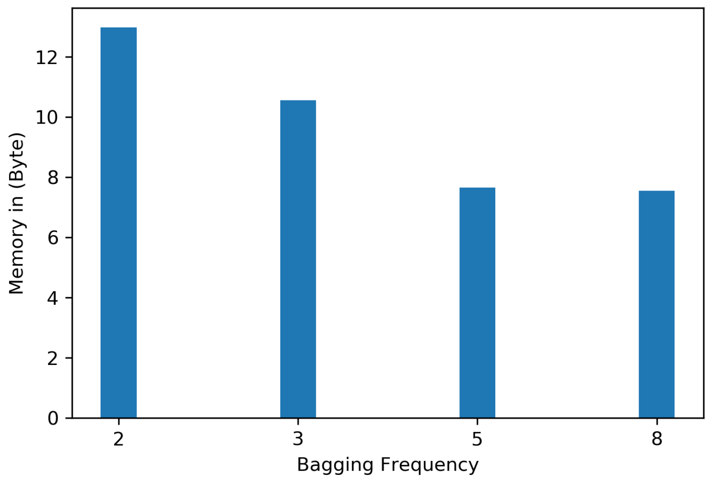

Figure 3.11: Effect of bagging frequency on memory with N-BaIoT.

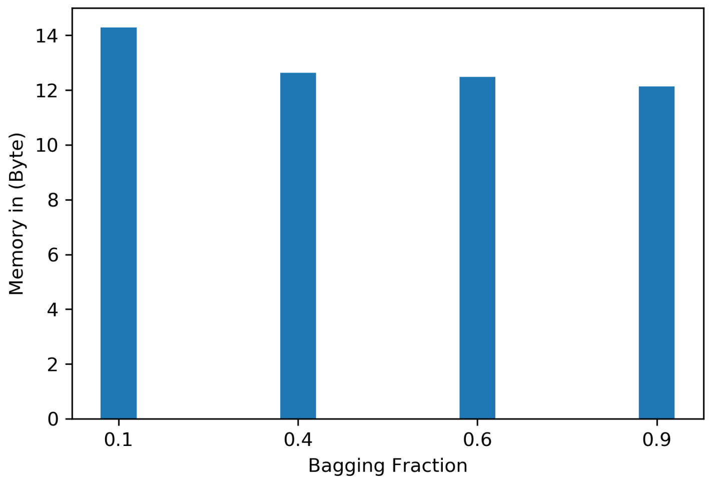

Figure 3.12: Effect of bagging fraction on memory with N-BaIoT.

The results suggests the selection of accurate value where resources are limited and needs to be utilized. Also, Figure 3.13, demonstrates how feature fraction facilitates memory resource consumption. The illustration in Figure 3.14 is the memory consumption against the regularization term. Also, Figure 3.15 shows the impacts of learning rate on memory. It demonstrated that smaller values of these hyperparameters facilitate memory saving. These results demonstrate the effects of varying turning parameters of LGBM on memory consumption as tested with the N-BaIoT dataset. The results suggest that setting bagging fractions to 0.4 can provide almost similar memory savings as 0.9. In addition, a

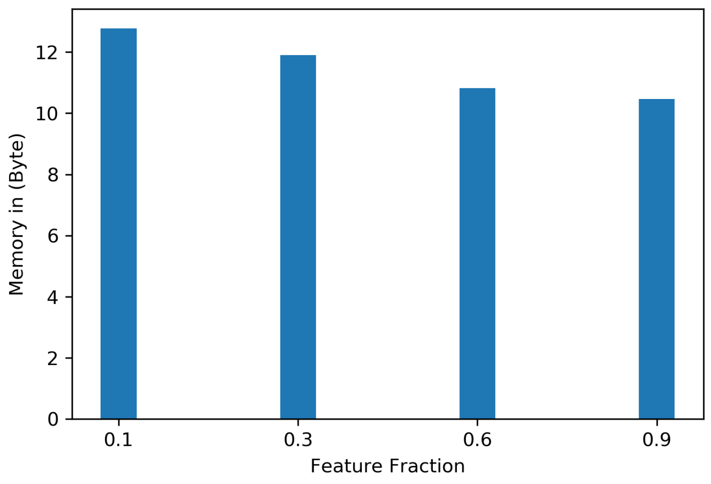

Figure 3.13: Effect of feature fraction on memory with N-BaIoT.

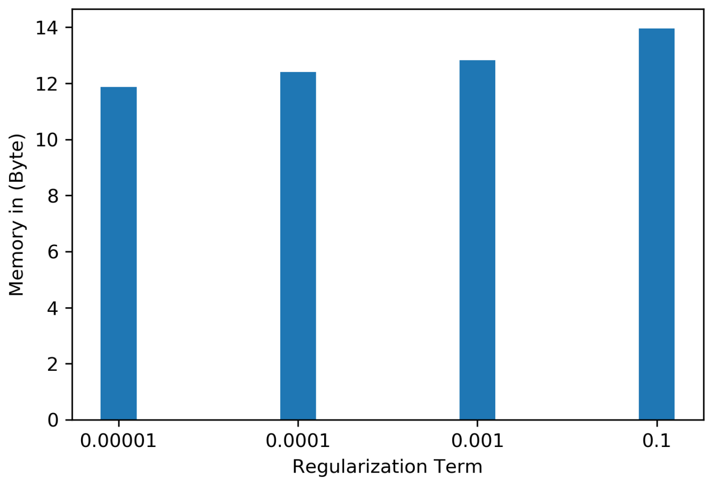

Figure 3.14: Effect of regularizer on memory with N-BaIoT.

lower value of the constraint number of tree leaves can reduce memory consumption. In our case, we find out that settings the number of tree depths to 2, enables more resource savings without degrading the prediction accuracy of the LGBM model. This is useful in training resource-hungry boosting algorithms to be suitable for utilization in a resource constraint environment.

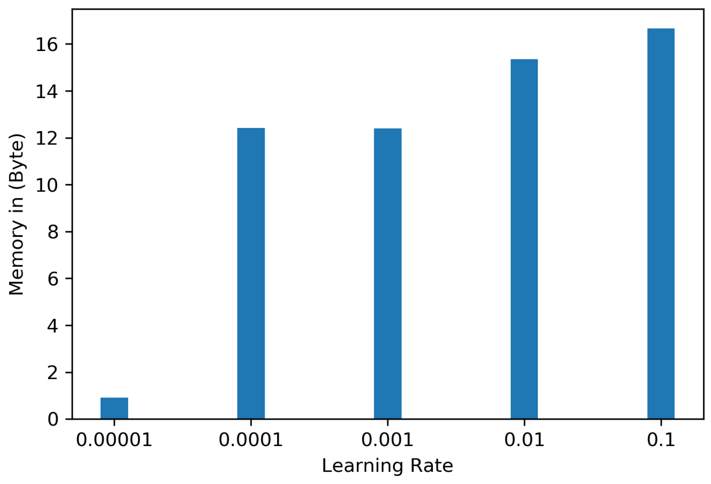

Figure 3.15: Effect of learning rate on memory with N-BaIoT.

3.2.4 Summary

In this chapter, we examined the feasibility of reducing the computational cost of training and testing ML algorithms for IoT cyber security monitoring. For this purpose, we employed carefully selected feature reduction methods, particularly the PCA and GI techniques. For proof of concept, we trained an AIS algorithm with a reduced data feature set of the KDD-99 and N-BaIoT cyber security datasets. The highest computational resource savings of memory and time usage occurred while using 10 PCA components of the N-BaIoT data. With this, it saved 80.08% and 80.10% of memory footprints in training and testing compared with the full data features. Regarding the running time, with 10 PCA components, the savings are 52.94% and 48.35%, respectively. However, the number of PCAs to be selected can be varied according to the dataset. The results demonstrate the influence of PCA over GI in terms of computation cost savings for running resource-hungry ML algorithms. These results motivate further explorations of various techniques to optimize ML algorithms for efficient and effective IoT attacks with better or state-of-the-art accuracy. The increasing number and complexity of IoT devices motivate the development of an efficient, and effective security protection system. We proposed such an approach that utilized LGBM with optimized hyperparameters to lower computational costs for resource constrained IoT devices. This is a requirement to reduce the computational expense of running traditional ML methods for IoT security monitoring. The proposed technique is efficient and useable for IoT resource consumption reduction. It outperforms the conventional LGBM model tested with the initialized hyperparameters and the grid search technique. It is better than the five employed boosting algorithms tested for effective attack detection and minimal resource consumption. In comparison with the SGB algorithm, it reduced the processing time and memory consumed during testing by 88.09% and 99.70% for each sample.

Chapter 4

Effective and Efficient Deep Learning for IoT Security Monitoring

Based on the work from the previous chapter (chapter 3) and consideration of the heterogeneous, complex security patterns and multidimensional nature of cyber security network traffic datasets [158] and DNN models can capture these complexities better than traditional ML models [159]. In this chapter, we want to investigate how to optimize DNN models for resource-efficient security monitoring in IoT networks. Looking at the promising success of DNN in various fields, they can be used to build a model for cyber security monitoring (see section 2.4). This is because proposing an accurate detection model is of practical interest for IoT cyber security as IoT technology is not yet mature and fully secured. However, building appropriate DNN-based IoT security solutions requires a large amount of training data. Such a procedure is computationally intensive and requires a lot of resources to provide effective attack detection. Because of that, DNN security solutions designed to deploy on general-purpose computing devices(e.g. Laptop, Desktop) may not be appropriate in a resource-constrained environment. Therefore, we are more concerned to build an efficient and effective DNN security mechanism for IoT devices. To this end, we present a framework in this chapter that can develop a suitable DNN-based method for the security monitoring of resource-constrained environments such as IoT. In our investigations, we use FCNN based on its performance in the context of cyber security [110], along with the IoT benchmark datasets described in Table 4.1 and exploit FCNN's optimization algorithm to obtain the REDNN version from the FCNN. The resulting REDNN demonstrates better classification performance with savings in execution time and memory footprint that can meet the minimum IoT resources requirements described in chapter 2 and address the limitations of previous study in chapter 2 (see Table 2.4). In addition, it can accurately detect attacks from each dataset used in our experiments.

4.1 Deep Learning for IoT Security

Based on the typical FCCN training described in Algorithm 1 as the core in subsection 2.3.7 and Algorithm 6 with an estimation of memory and time consumption, we slightly customize its training optimization to obtain REDNN. The memory and time are profiled after the completion of model training. These are the model training computational cost. We described this memory profiling tool in the implementation described in section 4.2. Algorithm 6 served as a baseline procedure for proposing its optimized counterparts (REDNN).

Algorithm 6 Baseline FCNN training

```
    Input:  Labelled data $\mathcal{D}_{tr}$, Number of iteration $\mathcal{T}$, Batch size $\mathcal{S}$
    Output: Baseline model $\mathcal{M}_n$
 1: function BASE($\mathcal{D}_{tr}$[ ])                                   ▷ Training baseline model
 2:     for $i = 1$ to $\mathcal{T}$; do
 3:         Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m)\} \subset D_{tr}$   ▷ $B \leftarrow |D_{tr}|//\mathcal{S}$
 4:         $F_p(B)$                                                       ▷ Forward propagation
 5:         $\mathcal{E}_i \leftarrow L$                                   ▷ $L$ = Base loss
 6:         $B_p$(B)                                                       ▷ Backward propagation
 7:         Compute gradients for parameters update
 8:         Estimate $m_i$                                     ▷ Execution memory at epoch $i$
 9:         Estimate $t_i$                                       ▷ Execution time at epoch $i$
10:         $\mathcal{M}_n$ = Trained model that estimate $\mathcal{E}_i, m_i, t_i$
11:     end for
12:     return $(\mathcal{M}_n, m_i, t_i, \mathcal{E}_i)$
13: end function
```

$$E = \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1 + w_j^2/w_0^2)} \tag{4.1}$$

To protect IoT networks from malicious attacks we need an effective and efficient model. This is the model with minimal training and testing computational cost and better or state-of-the-art accuracy. However, the procedure of finding an effective and resource-efficient DNN model can be a challenging task [160]. This is due to the various parameter requirements in designing and building the desirable architecture, particularly with complex and multidimensional datasets, especially the IoT traffic data produced from many commercial devices consisting of resource constraints, lower memory and processor. We propose such a framework that manipulates and optimizes an FCNN version of DNN to yield a compact classification model (see Figure 4.1). In our procedure, we utilize the baseline $\mathcal{M}_n$ model in Algorithm 6 to propose the optimized REDNN model. As demonstrated in Algorithm 7 the optimized model adopts micro-batching [161, 162] for efficient model building. The function procedure requires $\mathcal{D}_{tr}$ in mini-batch and micro-batch forms and iterates $\mathcal{T}$ times repeatedly to return the efficient M_e representing the REDNN model. The optimization process utilizes penalty function (weight elimination) [163] represented by E in Equation 4.1 with a weight threshold parameter w_0. The expression in line 7 of Algorithm 7 is responsible for pruning the network model weights to reduce its architectural complexity by incorporating weight elimination into the baseline loss. This procedure is useful in distinguishing the sets of relevant weights that can enable efficient model learning from the irrelevant ones, particularly the insignificant large weights of the baseline $\mathcal{M}_n$ model that is initialized within [0,1] at implementation. In the process, weights values W greater than w_0 can yield a complexity cost closer to 1 and require regularization using the penalty parameter λ. This is important to reduce the complexity of the model to enable faster training. As we are more concerned with a less complex, efficient and effective model building that can retain its performance, therefore, we consider the set of parameters that can give a training error $\mathcal{E}_j$ lower than the binary cross entropy loss $\mathcal{E}_t$ described in Equation 2.7 of section 2.3.7. The most important parameters are the w_0 and λ which are the threshold that controls the learning of the model while reducing its architecture. In line 10, we compared the regularization error $\mathcal{E}_j$ with the initialized error $\mathcal{E}_t$ before the regularization. This is to examine the convergence rate of the model during each epoch iteration. Based on the outcomes of line 10, relaxation of the λ value using the $\triangle\lambda$ occurred in line 11. The memory footprint and execution time are estimated in lines 12 and 13 and compared with the initialized values in lines 14 and 15 from line 6. This process is determined to find a model architecture with a faster convergence rate and minimal memory and time requirements in training. Due to the regularization in lines 10 and 11 of Algorithm 7, the returned REDNN model is less complex.

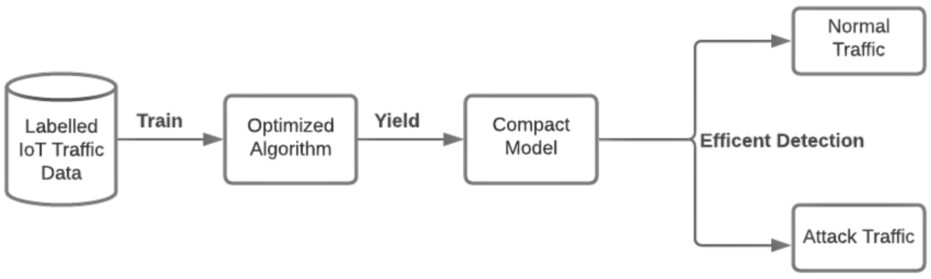

Figure 4.1: Effective IoT attack detection framework.

Algorithm 7 Proposed algorithm to obtain REDNN

```
    Input:  Penalty term $\lambda$, ($\mathcal{D}_{tr}, \mathcal{T}, B$, in Alg. 6)
    Output: Efficient model $\mathcal{M}_e$
 1: function EFFICIENT($\mathcal{D}_{tr}$[ ])
 2:     for $j = 1$ to $\mathcal{T}$; do
 3:         Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$        ▷ $B \subset \mathcal{D}_{tr}$
 4:         $F_p(M)$                                              ▷ Forward propagation
 5:         $\mathcal{E}_t = L$                                           ▷ L = Initial loss
 6:         $m_t$, $t_t$                   ▷ $m_t$, $t_t$ estimated memory and time using $\mathcal{E}_t$
 7:         $\mathcal{E}_j \leftarrow \mathcal{E}_t + \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$
 8:         $B_p$(M)                                             ▷ Backward propagation
 9:         Compute gradients for parameters update
10:         if ($\mathcal{E}_j \leq \mathcal{E}_t$) then
11:             $\lambda = \lambda + \triangle\lambda$
12:             Estimate $m_j$                          ▷ Execution memory at epoch $j$
13:             Estimate $t_j$                            ▷ Execution time at epoch $j$
14:             if $m_j \leq m_t$ then
15:                 if $t_j \leq t_t$ then                    ▷ $m_t$ = Efficient memory
16:                     $m_t = m_j$
17:                     $t_t = t_j$                              ▷ $t_t$ = Efficient time
18:                     $\mathcal{M}_e$ = Trained model that estimate $\mathcal{E}_j, m_t, t_t$
19:                 end if
20:             end if
21:         end if
22:     end for
23:     return ($\mathcal{M}_e, \mathcal{E}_j, m_t, t_t$)
24: end function
```

4.2 Evaluation

This section describes the evaluation criteria of the baseline FCNN and REDNN models. It also presents the datasets used in building the models. The N-BaIoT [70], Kitsune [74], WUSTL [76] datasets described in section 2.3.9 were employed in our experiments in this chapter. As we are interested of using device centric datasets which are simulated from IoT devices, other than IP based datasets (unsw), so we employed few of them. This is to assess our algorithm performance in device centric settings. A brief description of each dataset is as follows.

Data Preprocessing

The choice of these datasets allows frequent model training and rigorous evaluation. These datasets represented by numeric traffics flow are highly in-balanced suitable for IoT security investigations. Each utilized dataset is separated into 80% for training and 20% testing samples. Data input vectors are normalized using the unity-based normalization feature scaling.

Experimental Setup

We used Python 3.76 on a desktop computer with the specifications described in subsection 3.1.1 to build each model. For profiling model memory consumption, we utilized the integrated memory usage [149]. At training, parameters remain constant to enable a fair comparison. This applied to the baseline FCNN model, optimized REDNN and adversarial process.

Implementation Details

FCNN and REDNN Models Design. For building the generic sequential (dense) FCNN and REDNN models with each dataset, we utilized the scientific NumPy python module [164]. This enables the building of an in-depth DNN model without any library. This is good to understand underlying concepts and explore the internal operations within the network. Each model consists of an input layer, four hidden layers and an output layer as illustrated in Table 4.1. Regarding the topology selection against each dataset, we utilized the best-run Hyperas modules [165]. With that, we can select the best topology configurations against each dataset. The topology selection can minimize operations while maximizing the performance metrics. These are the requirements for the task of binary classification. The architectural settings remain identical for evaluating the baseline FCNN and the proposed REDNN model. Table 4.1 describes the models

Table 4.1: Topology and distribution of normal and attack for each device data.

Device	Normal	Attack	Inputs	Output	Topology
Danmini Doorbell	49,548	968,750	115	1	128-128-128-128
Ecobee Thermostat	13,113	822,763	115	1	32-64-64-16
Ennio Doorbell	39,100	316,400	115	1	64-128-128-64
Philips B120N10	175,240	923,437	115	1	128-128-128-128
Provision PT-737E	62,154	766,106	115	1	128-128-128-128
Provision PT-838	98,514	729,862	115	1	128-128-128-128
Samsung SNH-1011-N	52,150	323,072	115	1	128-128-128-128
SH XCS-1002-WHT	46,585	816,471	115	1	128-128-128-128
SH XCS-1003-WHT	19,528	831,298	115	1	128-128-128-128
Kitsune	121,621	642,516	115	1	128-128-128-128
Wustl	6,566,438	471,545	6	1	128-128-128-128

topology against each tested data.

For training, a mini-batch gradient descent optimizer with momentum was used. Random initialization of weight and bias parameters are within [0,1]. The baseline and optimized training procedure utilized $lr = 0.001$ across each dataset except the Ecobee and Ennio devices data with a different topology that used $lr = 0.0001$. Both FCNN and REDNN utilized a momentum value of 0.001. We used 0.01 values for λ, $\triangle\lambda$ and threshold w_0 [166] with 4 micro-batches to build the REDNN model. Models are trained in 128 batches within the 100 epochs for accuracy to converge. Binary cross entropy was used for calculating loss function. The activation function used in the input layer is ReLu [167] and Sigmoid for the output layer. For efficient hyperparameter selection, we utilized an automatic optimizer search module [168]. This technique required a range of values for each hyperparameter to be tuned to return an efficient combination. We used Numpy.float16 modules to implement the float 16 precision for the baseline and optimized model. A TensorFlow Core version (v2.8.0) is used for building the Keras and TensorFlow deep learning models. The TensorFlow Lite (TFLite) converter module is used to build the TFLite deep learning model. The implementation used a fair comparison with Numpy (FCNN and REDNN) as both the Keras and TFLite models are trained in 128 mini-batches using a Stochastic Gradient Descent (SGD), at 100 epochs iterations. For the linear SVM, Adaboost and GB models, we utilized the Scikit-learn [148] machine learning python framework. For exploration and reproduction purposes, the codes used for this study is accessible at [169]. Regarding the TFLite experimentation, we include both the Jupyter notebook file and the python script in the GitHub repository [169].

4.3 Results (Effectiveness and Resource Efficiency)

This section discusses the experimental results. It details the evaluation comparison of the REDNN and optimized FCNN models across datasets.

REDNN Model Resource Usage

Figure 4.2 presents the execution time in seconds for training each model. Since we are considering various datasets, we reported the cumulative memory usage by each model against each tested dataset. As demonstrated, the REDNN model is more efficient with minimal cumulative execution time against each dataset. This suggests its less computational expensive nature.

Figure 4.3 illustrates the CPU performance (1/*execution time*) for training each model. The values in the y-axis of Figure 4.3 represent the clock rate (frequency) which indicates how fast the CPU can run each model against each tested dataset. The performance comparison is over one CPU with specific architecture. In each case, a higher value means the CPU can performs certain operations faster. As expected, the REDNN model is more efficient with a better CPU performance frequency value against each dataset. These resources saving capabilities make it an appropriate model for intrusion detection in IoT network environments. The results agree with the RQ2 of this thesis while indicating that an existing DNN model can be trained to be resource-efficient better than its baseline counterparts.

To further investigate RQ2 based on the motivational results of the resource savings, we present measured testing results of the eleven IoT datasets runs with the FCNN and REDNN models in Table 4.2. In each case, the testing memory consumption of each algorithm is profiled in MB against each tested dataset. The reported memory results are for the cumulative records of each dataset. As demonstrated, the REDNN model can operate with a lower memory footprint. It can process the Wustl and Ennio Doorbell datasets with 98.84% and 77.63% of memory savings than the baseline FCNN. These resource minimizations make it a better choice for IoT security monitoring. It indicates its less complexity and faster detection capability. The results indicate that it is possible to reduce the computational resources of the FCNN model without degrading the testing accuracy. This is the case of the nine device subsets of the N-BaIoT dataset, and the FCCN network architecture utilized to build the REDNN model. As such, the regularized REDNN model can provide similar or better detection accuracy with its FCNN counterparts based on the regularization threshold in line 10 of Algorithm 7. We

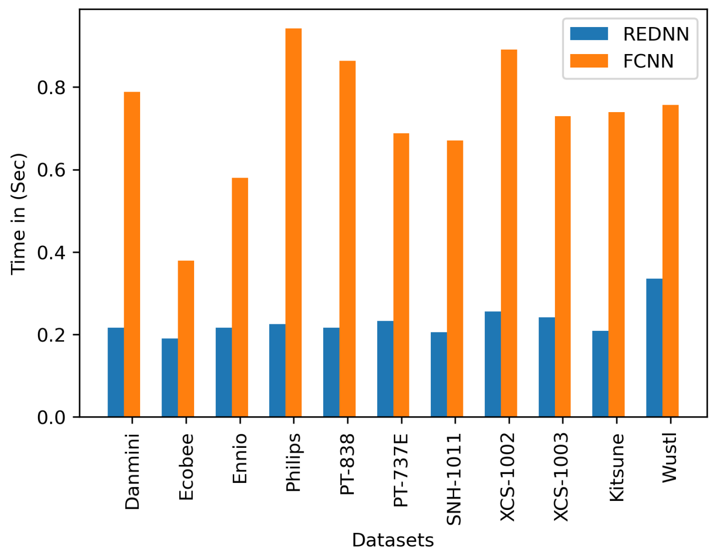

Figure 4.2: Model training execution time against utilized datasets (cumulative).

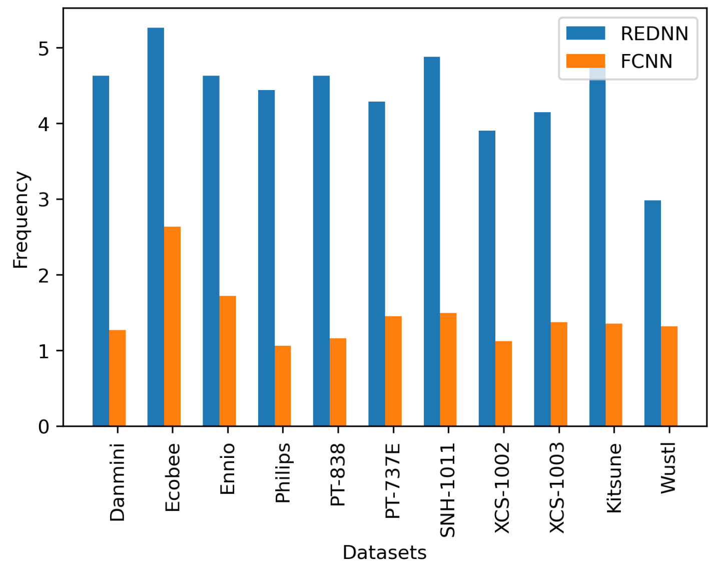

Figure 4.3: Model training CPU performance against utilized datasets (cumulative).

demonstrated an instance which the REDNN provides better detection accuracy than its counterparts in chapter 5 and 6. These results answered the RQ2 of this thesis and agree with our argument that FCNN models can be optimized to reduce the computational complexity without degrading their detection capability.

Table 4.2: Testing memory footprint (cumulative).

Dataset	Model	Mem (MB)	Mem save (%)	Test acc (%)
Danmini Doorbell	FCNN	3.742	N/A	95.11
	REDNN	1.555	58.44	95.11
Ecobee Thermostat	FCNN	2.804	N/A	93.36
	REDNN	1.277	54.46	93.36
Ennio Doorbell	FCNN	2.410	N/A	88.94
	REDNN	0.539	77.63	88.94
Philips B120N10	FCNN	3.738	N/A	84.08
	REDNN	1.731	53.71	84.08
Provision PT-838	FCNN	3.031	N/A	88.07
	REDNN	1.266	58.23	88.07
Provision PT-737E	FCNN	3.008	N/A	92.52
	REDNN	1.285	57.28	92.52
Samsung SNH-1011-N	FCNN	2.598	N/A	86.07
	REDNN	0.582	77.60	86.07
SH XCS-1002	FCNN	3.004	N/A	94.65
	REDNN	1.320	56.06	94.65
SH XCS-1003	FCNN	3.145	N/A	97.72
	REDNN	1.305	58.51	97.72
Kitsune	FCNN	2.726	N/A	84.09
	REDNN	1.168	57.15	84.09
Wustl	FCNN	491.6	N/A	94.26
	REDNN	5.711	98.84	94.26

REDNN Model Performance Comparison

The descriptions in Table 4.3 compare the performance evaluation of REDNN as implemented against various state-of-the-art technology frameworks (libraries). This demonstrates the potentiality of the optimized REDNN of savings resources in different experimental platforms. At training, REDNN demonstrates better memory footprint and time savings against each data record. It saves 99.86% and 99.99% of training time and memory footprint than the baseline model trained with Keras as computed based on the reported values in column Train time $((0.0196/13.189 * 100) - 100)$ and Train mem $((0.1388/3127.5 * 100) - 100)$ from Table 4.3, respectively. As compared with the converted FCNN TFLite model, the REDNN demonstrates better memory usage. The reason can be the TFLite model inherits the default Keras parameters during the model conversion. As a result, the conversion produces a lighter version of the Keras model.

Table 4.3: Training performance evaluation across frameworks with Provision PT-737E dataset (per record).

Procedure	Train time (ms)	Train mem (B)	Test set acc (%)
FCNN-Keras	13.189	3127.5	92.52
FCNN-TFLite	0.1605	372.29	92.52
FCNN-Numpy	0.0571	16.933	92.52
REDNN-Numpy	0.0196	0.1388	92.52

However, the quantized optimized TFLite model consumes fewer resources. It required 0.010 ms of training execution times and 0.0060 B of training memory footprint. This is due to the computations with tf.float16 low precision [170] at training while optimizing the converted TFLite model [171]. However, in some cases, low precision can cause numerical issues [170]. This can leads to accuracy performance degradation with some datasets [172]. Because of that we implement each framework in 32 bits and compare their performance in Table 4.3. This is useful to investigate the resource savings without the low precision integration. As presented, the significant training resource-saving of the optimized REDNN model can be useful for on-device learning. These motivational results serve as an avenue for the utilization of the optimized REDNN model to evaluate the hypothesis stated in RQ3.

In Table 4.4, we show the testing resources (memory and time) consumption by each model against utilized frameworks. Regarding the processing times of each framework against each data record, the NumPy implementation is faster. REDNN can process IoT data efficiently at a slightly higher rate than the baseline FCNN model runs on the same framework. The TFLite model is more efficient than the Keras model but slower than Numpy (FCNN and REDNN) models. Overall, REDNN saves 4.309 %, 69.81% and 80.55% of processing times than FCNN, TFLite and Keras models, respectively. This interesting result demonstrates the resource-efficient nature of our training procedure with Numpy better than other state-of-the-art training optimization methods. Therefore, compared to the currently available state-of-the-art methods, it can be an appropriate procedure for training and building an effective model in a resource-constrained environment.

For the memory consumption in column (Test mem), REDNN demonstrates better savings with each data record. The reduction is by 78.91%, 80.12% and 98.51% of memory footprint than FCNN-Numpy, FCNN-TFLite and FCNN-Keras models, respectively. The

Table 4.4: Testing resource consumption across frameworks with Provision PT-737E dataset (per record).

Procedure	Test time ((ms)	Test mem (B)	Test set acc (%)
FCNN-Keras	2.3522	512.64	92.52
FCNN-TFLite	1.5155	38.533	92.52
FCNN-Numpy	0.4781	36.317	92.52
REDNN-Numpy	0.4575	7.6606	92.52

TFLite more resources consumption is due to the data type conversion during prediction [171]. The conversion can increase the execution time and memory [173] as demonstrated in Table 4.4. The higher resources (memory and time) consumption of the TFLite at the testing stage is a limitation for effective IoT attack detection. As REDNN demonstrate minimal testing resource consumption, it can be utilized as an effective mechanism for IoT security monitoring.

Table 4.5 presents measured results that compare the performance of the REDNN model against state-of-the-art techniques using the PT-737E dataset. The reported results are the computational resources required by each model to process each record of the PT-737E dataset. In each case, the REDNN model demonstrates better memory and time resource savings. At training, it saves more than 99.99% and 99.80% of execution time and memory footprint than the SVM model based on the memory and time usage reported in Table 4.5. The reason is that SVM can be a more computationally expensive ML algorithm with complexity in training large datasets [174]. This makes it to be more resource hungry than the Adaboost and GB decision tree models. As DNN outperforms conventional ML models, the FCNN and REDNN results are as expected. The results suggest that DNN model optimization can create an efficient method with more resource savings than traditional ML methods. This is good in building a model in an environment with a multi-dimensional and large training dataset that requires significant resource savings.

For testing, the reduction is 99.91% and 99.17% of execution time and memory footprint against the SVM model as computed based on SVM and REDDN resource usages in Table 4.5. The reason can be SVM requires much processing time to classify the various traffic flows present in the PT-737E dataset [175]. The testing memory reduction remains promising across all tested models. For instance, as compared with the AdaBoost model, REDNN saves 82.49% and 44.66% of testing time and memory usage, respectively. This is

Table 4.5: Performance evaluation comparison on Provision PT-737E dataset (per record).

Model	Train time (ms)	Test time (ms)	Train mem (B)	Test mem (B)	Test set acc (%)
SVM	909.64	500.87	378.96	923.48	92.52
GB	32.621	0.2242	22.230	20.018	92.58
AdaBoost	31.212	2.6126	4.1910	13.842	92.47
FCNN	0.0571	0.4781	4.2333	7.9685	92.52
REDNN	0.0196	0.4575	0.0347	7.6606	92.52

good as the proposed REDNN can provide effective and efficient detection using minimal memory consumption.

4.4 Chapter Summary

This chapter served to evaluate the feasibility of using AI techniques in IoT security monitoring in a resource-efficient manner. In this context, we introduced a procedure to obtain an efficient and effective method (REDNN) by exploiting the training algorithm of the model. In building the REDNN, we utilized FCNN and proposed a model that maintains better performance for intrusion detection of features of the IoT network traffic. We demonstrated its performance through empirical evaluation using eleven datasets. The technique can accurately detect attacks from each IoT device's dataset. It demonstrates more efficient performance than its counterpart in terms of effectiveness and resource consumption. In addition, it outperforms all the conventional ML algorithms tested in terms of resource savings. At training, REDNN can save more than 99.99% and 99.80% of execution time and memory footprint as compared with the SVM model. With this trend of resource savings, it outperforms other state-of-the-art methods investigated in this thesis. In addition, it is better than a quantized TFLite model using a similar architecture and a model developed using the Keras framework. In each case, our training procedure is better, with the advantage of reducing computational resources while retaining the detection accuracy. The training and testing resource-saving results for each device model and accurate detection of attack samples depend on the topology settings and effective parameters optimization. The results suggest the potentiality of the DNN algorithm in providing an effective and efficient solution for IoT network intrusion detection tasks. This is useful in finding an optimal model for different datasets in IoT security monitoring or any other classification problem using DNN in general. An essential approach to this would be to enhance this study's results to investigate the impact of adversarial perturbations in the supervised and unsupervised scenarios for robustness and resilience testing.

Chapter 5

Robust Deep Learning for IoT Security Monitoring

Based on the success of training DNN for effective and resource-efficient IoT attacks detection discussed in chapter 4, this chapter presents a technical background to address the adversarial robustness of DNN utilized for IoT security monitoring. It then exploits each technique in detail to better understand their adversarial potential to attack the proposed REDNN, especially for the task of robustness testing in resource constrained environments of IoT. This is to further investigate the research hypothesis stated in RQ3.

5.1 Adversarial Robustness Implementation

The utilized documentation enables the development of generic adversarial attack methods using NumPy python modules. As the utilized datasets features are normalized within [0,1], each employed attacking technique except the semantic utilized an epsilon value scale within [0,1] incremented by 0.1. These scaling values can determine the proportion of perturbation data generated and their success rate in fooling various DNN models. For the PGD, an infinity norm value was used with 40 iterations to further examine the attack success rate. This method enables the production of a new set of testing data that can be evaluated against the baseline FCNN and REDNN models.

For crafting the FGSM and PGD adversarial samples, we utilized Algorithm 2 and Equation 2.15 described in section 2.5 of chapter 2 along with the cleverhans documentation [176].

Another perturbation procedure considered in this chapter is the data poisoning attacks with description in Algorithm 8. In this scenario, the data is poisoned by randomly flipping the labels (based on a random split of data features). The flipping procedure considers label modification for the attacks (0s) and benign (1s) samples. This is the all-labels modification technique that changes 1s to 0s and 0s to 1s, respectively. It is a non-targeted form of adversarial attack method that concentrates on both the benign and attack traffic classes. The rationale is to mislead the model by lowering its accuracy value to make it a weaker model. It achieved this by supplying the modified labels to each data feature while training the model. The trained model used testing data with correctly assigned labels for validation. We generate this form of attack by considering the training dataset. During implementation, the data samples are randomized before splitting to have a fair proportion of the attack and benign samples. All labels of the randomized samples are flipped based on the specified poisoning proportion, and to increase the chance of the success rate, we consider the rate to be from 0% - 50% by 5% increment. Each tested perturbation method used the preprocessed datasets described in section 4.2. These datasets are used to examine the success rate of each perturbation method to investigate REDNN resilience.

Algorithm 8 Label modification perturbation procedure

```
    Input:  $\mathcal{X}, \mathcal{Y}, n, p$, data, label, data length, percent
    Output: Poisoned data  $\{\mathcal{X}', \mathcal{Y}'\}$
1: for $t = 1$ to $n$; do
2:     if $t \in (1, p * n)$ then  ▷ Random samples selection as the dataset was randomized
3:         $y_t = 1 - y_t$                              ▷ Labels 0 and 1 modification
4:         $\mathcal{Y}' = \{(x_t, y_t)\}, t = 1 \ldots n$          ▷ Integrating label
5:     end if
6: end for
7: return $\{\mathcal{X}', \mathcal{Y}'\}$
```

5.2 Results and Discussion

This section discusses the experimental results of the perturbation techniques used to examine the adversarial robustness of REDNN and FCNN. It details the robustness evaluation comparison of the REDNN and optimized FCNN models across crafted samples generated from various datasets. The investigation results in this section can be used to validate the hypothesis stated in RQ3.

REDNN Model Robustness

In Table 5.1, comparison results between the performance of the REDNN and FCNN models against tested adversarial instances are presented. To ensure the convergence of the model and to have a fair comparison between the FCNN and REDNN models, the results are for 100 number of epochs iterations. In most cases, the REDNN provides higher test accuracy in detecting various adversarial perturbations based on regularization while optimizing the model. The regularization is to increase the robustness of the model. Using the SimpleHome XCS-1002-WHT device data, PGD reduces the accuracy performance of the FCNN and REDNN by 2.6% and 1.94%, respectively. With the Kitsune dataset, PGD affects the accuracy of the FCNN by 13.64%, while that of the REDNN by only 3.91%. These results demonstrate the capability of REDNN in resisting the most successful PGD perturbation technique with a significant attack success rate. The semantic attack is the weakest which shows no reduction in accuracy with each model. The reason can be altering the original samples by negation may not affect the accuracy with network traffic data rather than image format data [177]. For the noise attack, the REDNN provides better accuracy in many instances. The reason for not significant changes of accuracy with the tested data is the repeated iterative rounds that make the model to converges effectively. The regularized REDNN model can accurately detect several adversarial attacks in IoT network environments better than its counterparts. These results suggest that parameter optimization of an in-depth DNN model flowed by regularization can influence a robust detection of adversarial attacks in the context of IoT networks. This is demonstrated in Algorithm 7. After initialization of weights and bias parameters, the algorithm automatically adjusts the weight penalty parameter λ in line 11 of algorithm 1 to update the new loss define in line 7. As demonstrated, training with perturbed samples is not a requirement for effective and efficient detection in an IoT network environment. The results further indicate that REDNN can be robust against state-of-the-art perturbations. This result is good for answering RQ3 while proving the arguments that an optimized, regularized REDNN model can resist state-of-the-art adversarial attacks generated using IoT datasets better than its FCNN baseline counterparts. The results suggest further investigations of the REDNN adversarial robustness comparisons with other state-of-the-art conventional ML models for better performance assessment.

Table 5.1: Models performance comparisons across datasets.

Dataset	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)	Sem. acc (%)
Danmini Doorbell	FCNN	95.11	95.05	93.99	95.11	95.11
	REDNN	95.11	95.10	94.57	95.11	95.11
Ecobee Thermostat	FCNN	93.36	93.36	92.97	93.36	93.36
	REDNN	93.36	93.36	93.36	93.36	93.36
Ennio Doorbell	FCNN	88.94	88.90	88.90	88.94	88.94
	REDNN	88.94	88.89	88.89	88.94	88.94
Philips B120N10	FCNN	84.08	83.27	80.81	84.05	84.08
	REDNN	84.08	84.07	83.54	84.08	84.08
Provision PT-838	FCNN	88.07	87.19	84.75	88.06	88.07
	REDNN	88.07	87.83	86.70	88.07	88.07
Provsision PT-737E	FCNN	92.52	92.49	91.10	91.57	92.52
	REDNN	92.52	92.51	91.47	91.87	92.52
Samsung SNH-1011-N	FCNN	86.07	85.33	83.12	86.05	86.07
	REDNN	86.07	85.94	85.32	86.06	86.07
SH XCS-1002-WHT	FCNN	94.65	94.65	92.05	94.65	94.65
	REDNN	94.65	94.65	92.71	94.65	94.65
SH XCS-1003-WHT	FCNN	97.73	97.69	97.24	97.73	97.73
	REDNN	97.73	97.70	97.28	97.73	97.73
Kitsune	FCNN	84.09	78.27	70.45	81.00	84.09
	REDNN	84.09	83.52	80.18	84.02	84.09
Wustl	FCNN	94.26	94.26	94.26	94.26	94.26
	REDNN	94.26	94.26	94.26	94.26	94.26

Robustness against number of epoch. Table 5.2 shows the effect of epoch variation on model robustness against the SH XCS-1003 dataset. At ten epochs, the adversarial accuracy of both models reduces, especially the baseline FCNN, which showed a significant accuracy degradation, particularly with the PGD attacks that reduce the accuracy of the FCNN and REDNN to 69.65% and 77.43%. The reason may be due to the regualarization behaviour of the REDNN model, that makes it robust against attacks even at lower epoch. At 100 epoch iterations, the resilience of each model against perturbation attacks is better. The accuracy values for both the REDNN and FCNN are more than 97%, while the REDNN demonstrates better detection of the PGD attacks. This is useful as the optimized model can save more resources while thwarting adversarial attacks with higher epoch iterations. The results indicate that the REDNN robustness against adversarially crafted samples is better than that of the FCNN counterparts. Therefore, REDDNN can be a well robust IoT security method.

Table 5.2: Effect of number of epoch against models performance with SH XCS-1003 dataset.

Epoch	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
10	FCNN	97.73	79.51	69.65	89.52
	REDNN	97.73	86.70	77.43	89.79
20	FCNN	97.73	86.35	77.07	93.86
	REDNN	97.73	86.70	77.43	94.08
40	FCNN	97.73	93.74	86.66	97.08
	REDNN	97.73	94.19	87.10	97.17
60	FCNN	97.73	96.48	90.72	97.63
	REDNN	97.73	96.84	92.09	97.69
80	FCNN	97.73	97.48	94.82	97.72
	REDNN	97.73	97.53	95.34	97.73
100	FCNN	97.73	97.69	97.24	97.73
	REDNN	97.73	97.70	97.29	97.73

The results address the hypothesis stated in RQ3.

Robustness with clipped perturbation samples. Table 5.3 compares models' performance with clipped and non-clipped adversarial samples against randomly chosen datasets. In each case, the performance of detecting FGSM and random noise is better with the clipped procedure compared with non-clip settings. The REDNN outperforms its baseline counterpart in detecting the PGD and FGSM, especially with the kitsune dataset. For thwarting the non-clipped FGSM adversarial samples of XCS-1003 device data, REDNN and FCNN accuracy decreased by 0.41% and 0.45%, respectively. With the same procedure to detect the random noise attacks against the Kitsune data, the accuracy of the FCNN and REDNN decreased by 4.86% and 0.93%. These results demonstrate the resilience nature of the REDNN with clipped and non-clipped adversarial samples. With these results, we can suggest REDNN as the model that can craft adversarial attacks that are generated using various implementations.

Robustness against model variation. Table 5.4 presents the performance of REDNN and FCNN using three hidden layer model variational architectures. Across each tested dataset, REDNN resists adversarial attacks better than its counterparts. As tested against the Danmini Doorbell dataset, PGD attacks lower the accuracy of the FCNN and REDNN by 9.18% and 7.23%, respectively. With the optimized four hidden layer

Table 5.3: Effect of clipping samples against perturbations method.

Dataset	Procedure	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
SH XCS-1003-WHT	Clipped	FCNN	97.73	97.69	97.24	97.73
		REDNN	97.73	97.70	97.29	97.73
	Non-clip	FCNN	97.73	97.24	97.24	97.56
		REDNN	97.73	97.29	97.29	97.58
Danmini Doorbell	Clipped	FCNN	95.11	95.05	93.99	95.11
		REDNN	95.11	95.10	94.57	95.10
	Non-clip	FCNN	95.11	93.99	93.99	94.79
		REDNN	95.11	94.57	94.57	94.98
Kitsune	Clip	FCNN	84.09	78.27	70.45	80.67
		REDNN	84.09	83.52	80.18	83.84
	Non clip	FCNN	84.09	70.45	70.45	75.81
		REDNN	84.09	80.18	80.18	82.91

model architecture, accuracy reduction is by 1.12% and 0.54% for the FCNN and REDNN model (see Table 5.1). As expected, these results demonstrate that the four hidden layer networks can enable better attack detection. As a result, a neural network model with few hidden layers may not stand robust against adversarial attacks. As demonstrated, REDNN can detect adversarial perturbations regardless of the hidden layers utilized in building the network architecture. The results suggests the advantages of REDNN in an IoT network environment that can be dynamic in terms of architectural settings and security mechanism requirements.

Table 5.4: Variational models perturbations evaluations across datasets.

Dataset	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
Danmini Doorbell	FCNN	95.11	91.43	85.93	93.78
	REDNN	95.11	92.93	87.88	94.45
Provsision PT-737E	FCNN	92.52	90.31	86.31	91.61
	REDNN	92.52	90.81	87.20	91.91
SH XCS-1002-WHT	FCNN	94.65	92.48	87.87	93.54
	REDNN	94.65	93.21	89.02	93.99
SH XCS-1003-WHT	FCNN	97.73	96.51	92.20	96.98
	REDNN	97.73	96.62	92.33	97.03
Kitsune	FCNN	84.09	75.73	70.02	81.72
	REDNN	84.09	81.56	77.65	83.88

Figures 5.1 and 5.2 show the impact of epsilon value against variational architecture in detecting PGD adversarial attack. In Figure 5.1, the FCNN and REDNN used three hidden layers. In Figure 5.2, both REDNN and FCNN models utilized four hidden layers to detect PGD perturbations. This is to assess the impact of epsilon parameters on the success of PGD adversarial attacks. In both scenarios, test set accuracy decrease with the epsilon value of 1.0. The REDNN model can provide an incremental accuracy value of 0.6 and 2.0 with three and four hidden layers while detecting the PGD attacks. The reason may be removing a hidden layer of a model may affect performance accuracy. In our case, each hidden layer has 128 neuron values.

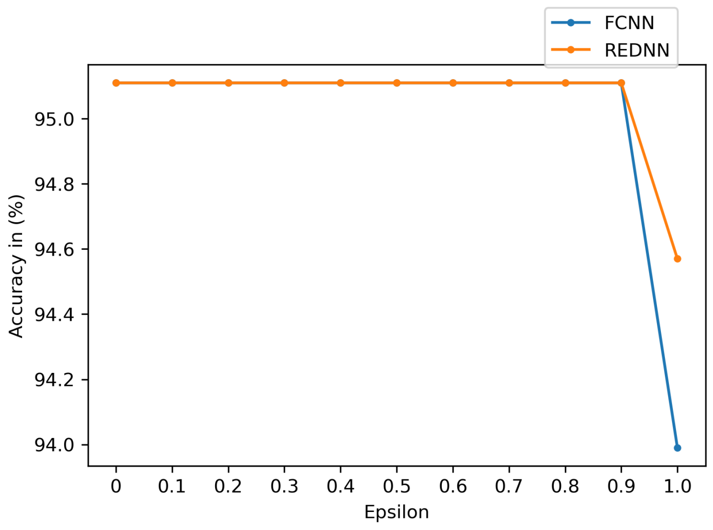

Figure 5.1: PGD test accuracy changes with epsilon for four layers architecture against the Danmini Doorbell dataset.

Figures 5.3 and 5.4 present the impacts of reducing the second hidden layer neuron of each model by the rate of 50% and 25% against resilience using the Kitsune dataset. In each setting, REDNN provides better detection accuracy against adversarial samples. As depicted in Figure 5.3, lowering hidden neuron values can affect the accuracy values of a model. It reduces FCNN and REDNN accuracy by 14.66% and 0.42%, respectively. For detecting PGD attacks using the 25% reduced neurons shown in Figure 5.4, FCNN and REDNN accuracy reduces by 24.52% and 5.26%, respectively. These results suggest that a significant reduction of hidden neurons affects the model's resilience against adversarial samples. In each scenario, REDNN stands to be more robust with topology variation than its counterparts. As a result, proper architecture selection can influence an efficient and effective identification of adversarial samples. This is due to the

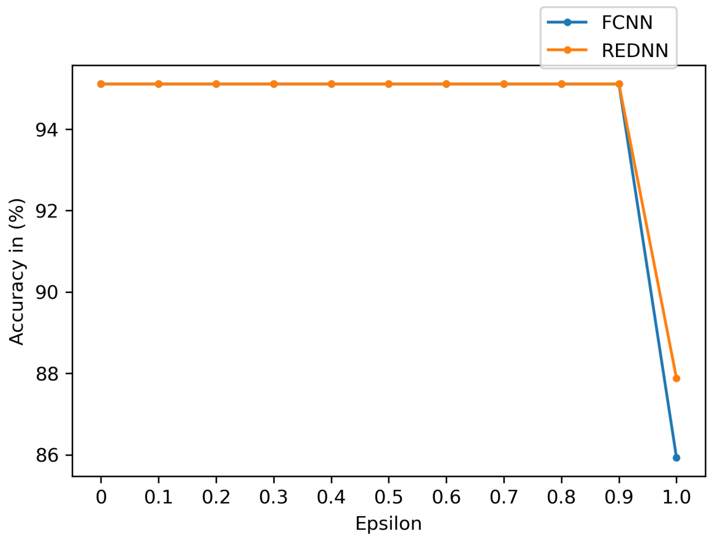

Figure 5.2: PGD test accuracy changes with epsilon for three hidden layers architecture against the Danmini Doorbell dataset.

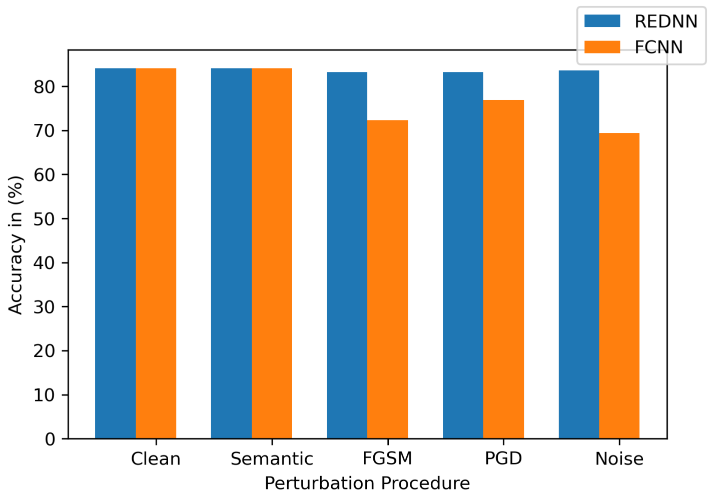

Figure 5.3: Accuracy changes with reduce hidden neuron by 50% against the Kitsune dataset.

architectural settings of the IoT network environment and the requirement of a robust mechanism that can detect attacks in respective of the topology utilized.

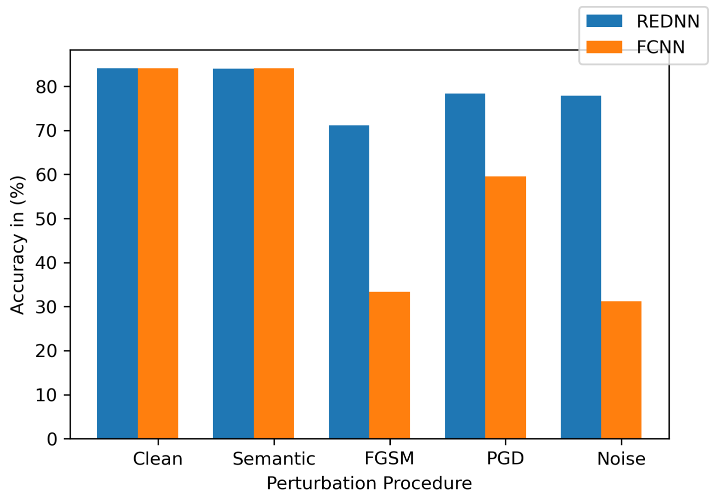

Figure 5.4: Accuracy changes with reduce hidden neuron by 25% against the Kitsune dataset.

We measured the results generated by poisoning the training data samples. A label flipping attack was integrated on each model using the kitsune and PT-737E device-centred dataset. The percentage of flipping rate considered is in the x-axis of Figure 5.5. Each model performs better with the modified data of the kitsune data (see Figure 5.5). They can thwart label poisoning attacks with 10% - 30% mislabelled training data as supported by the architecture of each model. Unlike Dunn et al. [178] that utilized 5% - 30% poisoned IoT data, we investigated further. With a 40% poisoning rate, REDNN significantly outperforms the FCNN model. For the PT-737E dataset result in Figure 5.6, the accuracy of the FCNN reduces significantly with 50% poisoning data. This behaviour may be due to the regularized property of the REDNN model [179]. Because of this, slighter changes in the training data may not affect the REDNN model. As demonstrated, the FCNN model is more sensitive to the 40% and 50% poisoning of the Kitsune and PT-737E datasets.

In addition to REDNN significant savings capability, it is more resilient against random noise attacks than each compared model (see Table 5.5). We later examined the effects of poisoning 50% of the training data with label modification (see Poisoned label column). The poisoning affects the robustness of the FCNN, SVM, GB and Adaboost model by lowering their accuracy to 9.55%, 7.48%, 10.01% and 11.05%, respectively. As demonstrated, REDNN indicates better resistance against labelled poisoned attacks. The results suggest that a stable and less complex model can defeat label poisoning attacks.

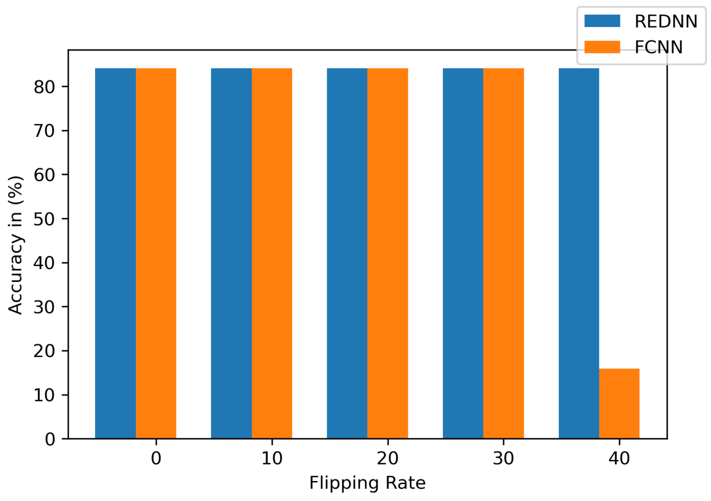

Figure 5.5: Accuracy changes with label flip against Kitsune dataset.

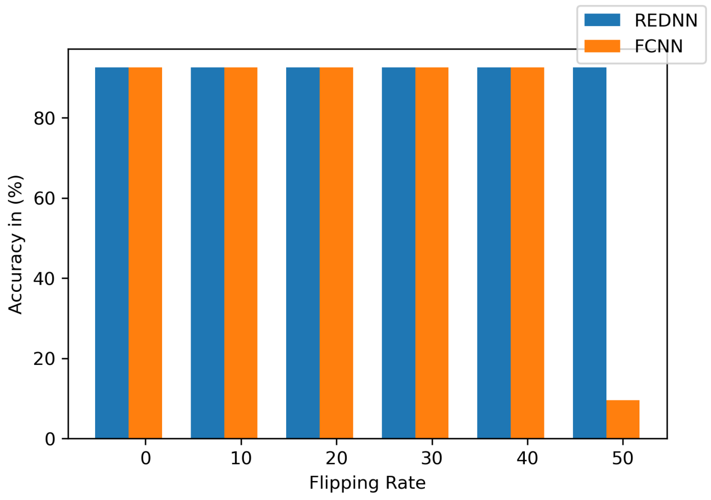

Figure 5.6: Accuracy changes with label flip against PT-737E dataset.

It further demonstrates the effectiveness and lightweight nature of the REDNN model. As a result, it stands appropriate for the task of IoT security monitoring.

Table 5.6 presents the model performance evaluated by test set accuracy, precision, recall and harmonic score (F1) while investigating the effects of FP16 integration on model resilience. Since the IoT datasets we considered are often imbalanced, we considered the test accuracy and other performance metrics. For further investigations, we used precision which considers the proportion of samples that are relevant within a predicted

Table 5.5: Performance evaluation comparison with Provision PT-737E dataset.

Model	Clean acc (%)	Noise acc (%)	Poisoned label acc (%)
SVM	92.52	70.89	7.48
GB	92.58	61.91	10.01
Adaboost	92.47	53.31	11.05
FCNN	92.52	91.57	9.55
REDNN	92.52	91.87	92.52

Table 5.6: Model resilience evaluation with kitsune dataset.

Attacks	Model	Accuracy (%)	Precision	Recall	F1 score
FGSM	FCNN	83.60	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
PGD	FCNN	82.34	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
Noise	FCNN	76.67	0.8412	0.8906	0.8652
	REDNN	83.73	0.8411	0.9944	0.9113

class and the F1 score that corresponds to the harmonic mean of precision and recall as other metrics to assess model performance. The utilized metrics used True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). The accuracy, precision, recall and F1 score are defined in Equation 2.10, 2.11, 2.12 and 2.13 (see section 2.3.10). This is to demonstrate the performance of the optimized model across different performance metrics.

The 16-bit floating-point low precision (FP16) implementation affects the robustness of the FCNN model, especially in defeating random noise attacks. As mentioned earlier in this thesis, low precision implementations are often chosen by machine learning engineers to make it run faster and use less memory during model training and testing, but at the cost of sacrificing overall accuracy. As demonstrated, REDNN indicates better resilience in thwarting each adversarial attack. The results suggest that FP16 integration had a minor influence on the robustness of the REDNN model. Because of this, REDDN is a more effective and robust IoT security monitoring technique than its baseline FCNN counterparts. With these results, we can conclude that REDNN is a robust, effective and efficient method, as tested against different procedures. The results demonstrate REDNN attack resilience capability even with integrated FP16 that can degrade model performance. The results discussed in this section are sufficient enough to prove the validity of the hypothesis stated in RQ3. Therefore, we address this hypothesis while exploring REDNN robustness against state-of-the-art perturbations methods and compare its performance over conventional ML methods.

5.3 Summary

This chapter discussed the investigation of exploring PGD, FGSM, Semantic, Noise and label-flipping adversarial attacks against FCNN and REDNN models for IoT security monitoring. It described the evaluation procedure of implementing each method while testing the adversarial robustness of various detection models intending to explore their resilience in detecting adversarial samples generated in an IoT environment. It presents the fundamentals of examining the FCNN and REDNN with other state-of-the-art ML methods in a robust scenario. As investigated, the REDNN demonstrates better resistance in thwarting adversarial samples than the FCNN counterparts. It can easily craft adversarial attacks that are generated in the clipped and non-clipped procedures better than the baseline counterparts, especially with the kitsune dataset at which REDN detects PGD attack with 83.53% and 80.18% in the clipped and non-clip training procedure. In addition, it can detect perturbed samples even with fewer epochs iterations, and the detection gets better at 100 epochs iterations. REDNN detection performance can scale through multiple model variations and DNN model network architectures. By comparing with the baseline FCNN and the SVM, GB and Adaboost conventional ML models, it demonstrates better resistance to label poisoned attacks. For instance, the label poisoned attack successfully degrades the accuracy of SVM, GB, Adaboost, and FCNN while REDNN accuracy remains unchanged. In addition, even with the FP16 integration, REDNN is better than FCNN, as it can be resilient to FGSM, PGD and Noise attacks in low-precision settings with better accuracy, precision, recall, and F1-score. In that case, it can detect the FGSM, PGD and Noise attack with more than 91% F1-score. The results demonstrate the performance advantage of REDNN even with low precision integration against adversarial attacks.

Chapter 6

Federated Deep Learning for IoT Network Security using REDNN

Following the feasibility of training FCNN to produce REDNN which is an effective and resource-efficient model in chapter 4, and its success for better resistance to various adversarial samples as discussed in chapter 5, this chapter considers further exploration of the proposed REDNN for on-device learning in FL settings using simulated virtual distributed nodes and realistic edge like decentralized devices. This exploration is due to the significant resource savings demonstrated by our optimized DNN training procedure and considerations of the distributed and resource-constrained nature of realistic IoT networks in practice. Unlike [147] FL scheme for detecting zero-day attacks in a simulated environment, our investigated FL training procedures considered virtually simulated distributed and an IoT-like network set-up with GB-BXBT-2807 edge-computing-like devices. Then, we propose a REFDL method that maintains state-of-the-art accuracy while reducing memory consumption in simulated and network testbed with GB-BXBT-2807 embedded devices experimental settings. This is useful to assess the feasibility of deploying a secure and efficient model in resource-constrained environments.

6.1 Baseline Federated Deep Learning (BFDL)

Following the explanation of FL in section 2.6, and considering its various usage in various areas, particularly in cyber security monitoring [180], its exploration in the aspect of IoT security monitoring can be beneficial [181]. This is due to the resource-constrained nature of IoT devices and FL's capability of preserving the on-device training data in proposing AI-based security mechanisms. This can be a core strategy for developing feasible security solutions that can detect attacks on IoT devices in real-time while granting privacy of each device's data. To this end, we illustrate the BFDL training procedure in Figure 6.1. It starts with creating a batch samples from the training data to train a local model. It later updates each weights of each local clients model before sending to the coordinating server for aggregation. Details of this procedure is described in in Algorithm 9.It uses the function BASE to train a baseline model using a SGD in FL settings. At each communication round, the function Device UPDATE in line 7 of Algorithm 9 is capable of distributing a master model to each client's subsets. Each client performs iterative rounds of gradient descent weights update with their local data and returns to the server in Algorithm 3. At this stage, the execution time and memory footprints are estimated based on lines 11 and 12 of Algorithm 9. These are the records of training resource usage at the device level after the local weights update. As expressed in line 17, computed model weights are returned to the coordinating server in Algorithm 3. The server is responsible for averaging the return weight for model aggregation.

Algorithm 9 Baseline BFDL training on each distributed node

```
    Input: Labelled data $\mathcal{D}_{tr}$, Iteration number $\mathcal{T}$, Batch size $\mathcal{S}$
    Output: Baseline model $\mathcal{M}_n$
 1: function BASE($\mathcal{D}_{tr}$[ ])                                  ▷ Training baseline model
 2:     for $i = 1$ to $\mathcal{T}$ do                                  ▷ for each local epoch iterations
 3:         Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m)\} \subset D_{tr}$   ▷ Mini-batch size ← $|S_{tr}|//\mathcal{S}$
 4:         $F_p(B)$                                               ▷ Forward propagation with $B$
 5:         $\mathcal{E}_i \leftarrow L$                                         ▷ $L$ = Base loss
 6:         $B_p$(B)                                               ▷ Backward propagation
 7:         function DEVICE UPDATE(($d, w$))                      ▷ Run on device $d$
 8:             $B_s \leftarrow$ (Split data $P_d$ into batches of size $S$)   ▷ $S$ is a local Mini-batch size
 9:             for batch $b \in B_s$ do
10:                 $w \leftarrow$ local weights update     ▷ device local weights update computation
11:                 Estimate $m_i$                          ▷ Execution memory at epoch $i$
12:                 Estimate $t_i$                          ▷ Execution time at epoch $i$
13:                 $\mathcal{M}_n$ = Trained model that estimate $\mathcal{E}_i, m_i, t_i$
14:             end for
15:         end function
16:     end for
17:     return $w$ to server in Alg. 3 ▷ Calls to coordinating server in Alg. 3 for weights averaging
18:     return $(\mathcal{M}_n, \mathcal{E}_i, m_i, t_i)$
19: end function
```

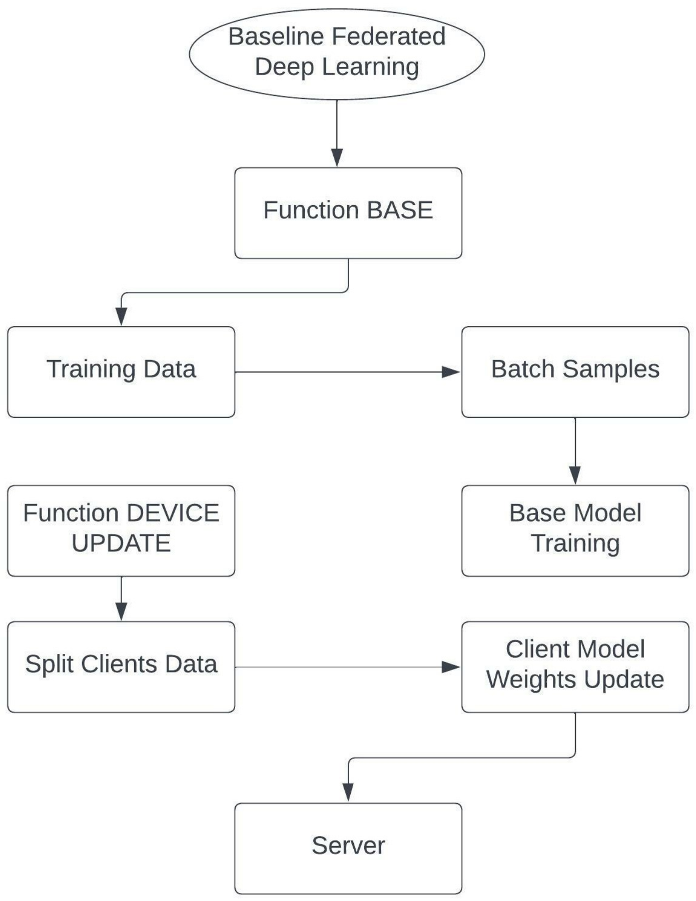

Figure 6.1: Baseline federated learning procedure.

6.2 Resource Efficient Federated Procedure

Training a resource-efficient DNN model for FL tasks can be a challenging task, especially in an IoT network environment. This is due to the FL communication rounds and DNN model parameters requirements in designing and building the desirable architecture [182]. The complexity of such an approach increases with multidimensional datasets. To demonstrate the proof of concept, we will use FedAvg as the core model (BFDL) with FCNN and CNN model variation against some IoT and non-IoT benchmark datasets and exploit its optimization algorithm to obtain the REFDL. To build REFDL, we used an optimized training procedure in Algorithm 10 in FL settings. For better performance, we utilized the set of model parameters that can produce a lower error based on line 7 of Algorithm 10. The function procedure in Algorithm 10 is responsible for computing and updating client device weights at each local epoch iteration before sending them to the

Algorithm 10 Proposed REFDL training on each distributed node

```
Input: Penalty term $\lambda$, ($\mathcal{D}_{tr}, \mathcal{T}, B, L$, in Alg. 9)
Output: Efficient model $\mathcal{M}_e$
1: function EFFICIENT($\mathcal{D}_{tr}[\,]$)
2:     for $j = 1$ to $\mathcal{T}$; do
3:         Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$        ▷ $B \subset \mathcal{D}_{tr}$
4:         $F_p(M)$        ▷ Forward propagation with $M$
5:         $\mathcal{E}_t = L$        ▷ Initialized loss
6:         Estimate $m_t$, $t_t$ Initialized memory and time based on $\mathcal{E}_t$
7:         $\mathcal{E}_j \leftarrow \mathcal{E}_t + \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$
8:         $B_p$(M)        ▷ Backward propagation with $M$
9:         function DEVICE UPDATE(($d$))        ▷ Run on device $d$
10:            $M_s \leftarrow$ (data $P_d$ in batches of size $M$)
11:            for batch $b \in M_s$ do
12:                $w \leftarrow$ local weights update        ▷ device local weights update computation
13:                if ($\mathcal{E}_j \leq \mathcal{E}_t$) then
14:                    $\lambda = \lambda + \triangle\lambda$
15:                    Estimate $m_j$        ▷ Execution memory at epoch $j$
16:                    Estimate $t_j$        ▷ Execution time at epoch $j$
17:                    if (($m_j < m_t$) $\wedge$ ($t_j < t_t$)) then
18:                        $m_t = m_j$        ▷ $m_t$ = Efficient memory
19:                        $t_t = t_j$        ▷ $t_t$ = Efficient time
20:                        $\mathcal{M}_e$ = Trained model that estimate $\mathcal{E}_j, m_{tr}, t_{tr}$
21:                    end if
22:                end if
23:            end for
24:        end function
25:    end for
26:    return $w$ to server in Alg. 3        ▷ Calls to Alg. 3 for model weights averaging
27:    return ($\mathcal{M}_e, \mathcal{E}_j, m_{tr}, t_{tr}$)
28: end function
```

coordinating server. In line 13 of Algorithm 10, the device model error is compared with the initialized error before model regularization in line 14. After this stage, in lines 15 and 16, the estimated computational memory footprints and execution time are compared with that of the initialized values in line 6 to return the minimal memory constraint produced by the client device model. Devices models with minimal resource consumption are returned to the coordinating server in Algorithm 3 together with their weights for model averaging. Then, the coordinating server can update the client model weights in a federated setting and performs weight averaging while returning the updated averaged weights for model aggregation. This process can reduce the client's communication time and computational complexity while building the resource-efficient aggregate model of REFDL. The memory and processor savings for each client device at each federated round and accumulating all these savings can lead to significant savings when the model is converged.

6.3 Evaluation

This section describes the evaluation procedure of the BFDL and REFDL methods. In addition to the utilization of the datasets and preprocessing procedure described in section 4.2 for techniques implementations, we further utilized the IoT-DDoS and MNIST datasets. IoT-DDoS consists of various captured traffics representing the DDoS botnet attacks and some portion of regular traffic [78]. The IoT-DDoS are distributed in nature, so ideal for the FL scenario. We consider 79,035 benign data and 398,391 attack data samples for empirical model evaluation. The MNIST handwritten digits dataset is a subset of the dataset from the National Institute of Standards and Technology [183]. The dataset consists of 60,000 training digits sample and 10,000 testing digits that are size-normalized, and each size contains 28*28 of 256 grey levels images. The MNIST dataset is suitable for model evaluations to assess its learning capability over the non-IoT cyber security dataset. This is useful to investigate whether the proposed model can learn complex patterns in other datasets. For this purpose, we utilized the MNIST dataset to assess REFDL performance against the image dataset.

Experimental Settings (Simulation)

To examine the behaviour of REFDL in FL settings, we start an initial investigation in a simulated network environment with virtual distributed nodes. This is to empirically test the capability of REFDL performance in different experimental settings and analyze its resource savings capability in non-realistic settings. We used a personal desktop computer with the experimental setup described in section 4.2. We utilized PyTorch version 1.4.0 [184] and PySyft version 0.2.9 [185] frameworks for the virtual on-device training. Pysyft framework simplifies the creation of virtual workers. We utilized these virtual workers to simulate the FL scenario for the BFDL and REFDL. These workers emulate real virtual machines and can run as a separate process within the same python program with their dataset. We further utilized a simulated testbed with WebSocket (WS) virtual and server workers that run on the same machine. In each case, our federation training procedure considered four clients virtual workers and a coordinating server worker receiving the computational updates from each virtual client worker model. Each federated client model consists of an input layer, four hidden layers and an output layer. Topology is selected against each dataset to minimize operations and improve the performance metrics [165]. The experimental settings considered are for the task of binary classification. For a fair comparison, the overall architectural settings remain identical for evaluating the BFDL and proposed REFDL technique. Table 6.1 presents the utilized model topology of each FL technique against each dataset. In addition to the implementation details described in section 4.2, both BFDL and REFDL use an SGD optimizer that is conventional for running FedAvg. Each federated model that utilized the PySyft virtual workers was trained in 128 batches within 4 local epochs iterations in 30 worker's communications rounds. Regarding the network simulated testbed, each model that used the WS workers was trained in 64 batches within 2 local epochs iterations in 50 workers' communications rounds. For fair experimental settings, the chosen values for local epochs and communication rounds remain identical in the realistic testbed experimental settings. After completing the client's model training, average weight values are sent to the coordinating worker acting as the execution server in the FedAvg algorithm. This worker aggregates the weights to update the global model. Codes for this implementation are made publicly accessible for exploration and reproduction purposes [186].

Table 6.1: Topology and distribution of normal and attack for each device data.

Device	Normal	Attack	Inputs	Output	Topology
Danmini Doorbell	49,548	968,750	115	1	83-128-128-83
Ecobee Thermostat	13,113	822,763	115	1	83-128-128-83
Ennio Doorbell	39,100	316,400	115	1	83-128-128-83
Provision PT-737E	62,154	766,106	115	1	83-128-128-83
Provision PT-838	98,514	729,862	115	1	83-128-128-83
Samsung SNH-1011-N	52,150	323,072	115	1	83-128-128-83
SH XCS-1002-WHT	46,585	816,471	115	1	83-128-128-83
SH XCS-1003-WHT	19,528	831,298	115	1	83-128-128-83
Wustl	6,566,438	471,545	6	1	26-128-128-26
IoT-DDoS	79,035	398,391	12	1	20-128-128-20

Experimental Settings (Testbed using GB-BXBT-2807 devices)

To test the efficient federated communication of the REFDL against BFDL in a testbed setting, we utilized the PySyft version 0.2.9 [185] python framework over a network with a client and server-class connected via a WS. Since PyTorch is a potential library for PySyft, we utilize it to build an edge computing FL training scenario for resource-constrained devices. The environmental settings are based on the client's server communication scenario in a distributed manner. It can support the building of simulated and realistic testbed settings. Figure 6.2 shows a high-level diagram of the testbed together with the utilized edge devices. In setting up the network realistic testbed settings, we considered 4 Gigabyte Brix (GB-BXBT-2807) ultra mini compact PC with a laptop (see Figure 6.3). Its processor is the 22nm Intel® Celeron N2807 while the memory capacity of the GB-BXBT-2807 (Ultra mini PC design – 0.69L(56.1x 107.6 x 114.4mm)) is pretty similar to that of Raspberry Pi version 4 model B and is SO-DIMM DDR3L at 1333MHz. However, the memory capacity can be extended up to 8 GB maximum size since we are implementing an on-device learning with this mini computer, we used its default memory to demonstrate the resource of our optimised method. We utilized these devices to represent the client in our decentralized wireless network testbed. This is to test the feasibility of creating a secure and lightweight FL model in realistic wireless network settings. The personal laptop represents the coordinating server in a wireless network to emulate low-frequency connections. The server is responsible for model weights aggregation and distribution to clients. Therefore, the communication workload is higher on the client side containing the edge devices than on the server machine. The installed Operating System (OS) in GB-BXBT-2807 clients is Ubuntu version 20.04.4 LTS. Each client contains an installation for the PySyft framework and its dependencies required for running the on-device federated learning. Federated network testbed implementations codes are publicly accessible [187].

For evaluating the simulated runtime and real execution time of BFDL and REFDL, experiments with four workers (Alice, Bob, Charlie and Jane as shown in Figure 6.3) were performed. A federated communication round of 50 is used, with two local epoch iterations, within a 64 mini-batch size. At each epoch iteration and federated rounds, a batch of samples is distributed to the client devices for local training and model pruning. The test batch sample size selection is 1000. A chosen $lr = 0.01$ was used for effective FedAvg SGD training. The utilized real-time models for each federated client contain an input layer and four identical hidden layers (128-128-128-128) with an output layer or layers as the case may be. The chosen architecture can support effective and efficient model convergence. To test the REFDL effectiveness and generalizability, we considered the CNN (DNN) variant in realistic settings with clients utilizing the MNIST image dataset. The CNN architecture utilized contains two convolutional layers (Conv-2). The first 2D convolutional layer requires one input to output 20 convolutional features using

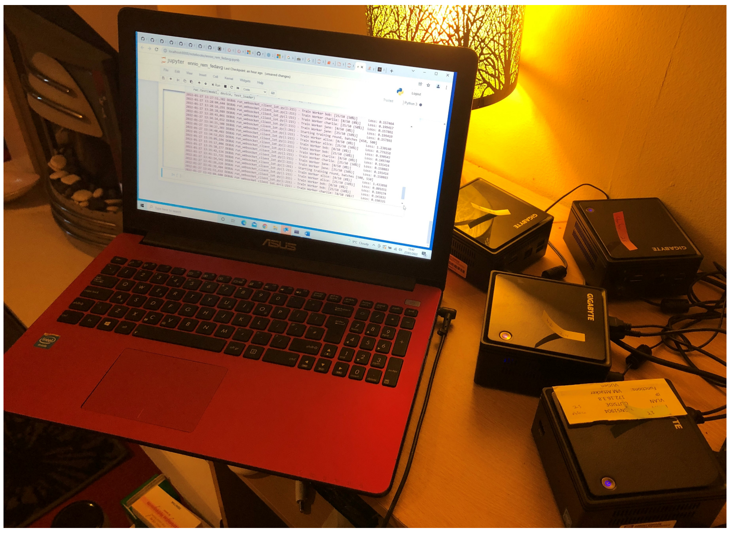

Figure 6.2: BFDL and REFDL model training testbed captured with gigabyte devices.

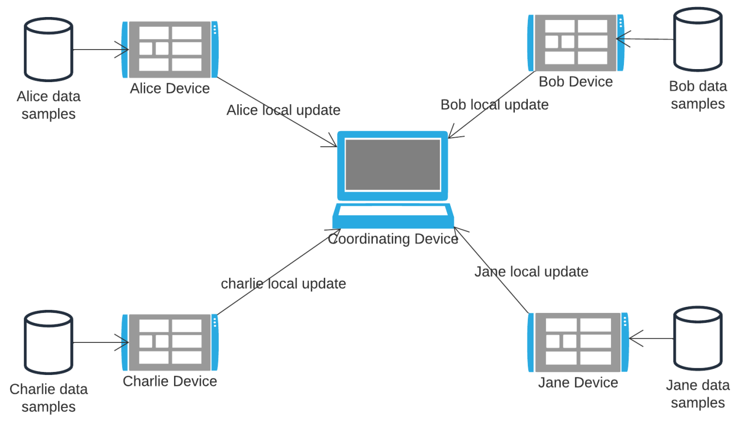

Figure 6.3: BFDL and REFDL model training testbed with gigabyte devices.

a 5 square kernel (1, 20, 5, 1). The second 2D convolutional layer requires 20 input layers to output 50 convolutional features using a 3 square kernel (20, 50, 5, 1). The architecture in the first real-time layer is (800 (4*4*50), 128) with (128, 10) in the second real-time layer. Max-Pool in 2d was run over the input image without a dropout utilization. These architectural settings remain identical for the BFDL and REFDL, however, in the FL training procedure, REFDL is optimized and regularized to reduce computational complexity and benefits from the resource-savings advantage. The fully connected hidden layers in the convolutional are similar to those described in Table 6.1.

6.4 Results and Discussion

This section discusses the experimental results for both the simulation using virtual workers and testbed settings with real edge-computing-like devices. It details the evaluation comparison of the optimized REFDL and baseline BFDL FedAvg models in simulation and testbed settings across datasets. Results from this section can be used to answer the hypothesis stated in RQ2. This is to support the argument that an existing DNN can be optimized to efficiently and effectively detect attacks on IoT networks in a federated scenario without accuracy degradation.

6.4.1 Experimental Results (Simulation)

We investigated the resource consumption for training BFDL and REFDL federated methods with nine utilized IoT datasets. As we considered various datasets in our experiments, Table 6.2 presents the cumulative averaged memory and time usage across each dataset for all virtual clients and server workers. As expected, the REFDL training procedure produces lower runtime and memory footprints against each tested dataset. This demonstrates the effectiveness and the capability of our proposed training method of saving resources across multiple federated client devices in a typical simulated federated setting. However, the accuracy for both REFDL and BFDL remained the same across each benchmark dataset. The reason can be the tested datasets are highly imbalanced with large number of testing records and considering the automatic pruning applied in lines 13 and 14 of Algorithm 10 based on the penalty function λ for the server execute. This is essential to regularized the loss function in line 7 of Algorithm 10 while maintaining its convergence behaviour. Refer to Table 6.4, Figure 6.12 and Table 6.8 for comparison with balanced dataset with minimal number of testing records.

Figures 6.4 and 6.5 show the percentage of memory and execution time saved by REFDL as reflected in Table 6.2. Across each dataset, REFDL saved more than 70% of memory units compared with the baseline counterparts. The results demonstrate a significant percentage of memory saving against each dataset and indicate the resource-savings advantage of REFDL. Regarding client processing runtime, REFDL is more efficient. It indicates less complexity, faster learning capability and effective performance behaviour over BFDL. These resources minimization make it a better choice for IoT security monitoring, especially for the task of on-device learning across various distributed resource-constrained edge devices.

Table 6.2: Federated model training memory consumption between REFDL and BFDL (cumulative).

Dataset	Model	Memory MB	Time minutes	Test acc %
Danmini Doorbell	BFDL	3.783	0.099	95.11
	REFDL	0.857	0.081	95.11
Ecobee Thermostat	BFDL	3.732	0.091	93.36
	REFDL	0.815	0.071	93.36
Ennio Doorbell	BFDL	4.147	0.090	88.94
	REFDL	0.805	0.074	88.94
Provision PT-737E	BFDL	3.463	0.092	92.52
	REFDL	0.853	0.077	92.52
Provision PT-838	BFDL	3.423	0.085	88.07
	REFDL	0.814	0.074	88.07
Samsung SNH-1011-N	BFDL	3.783	0.099	86.06
	REFDL	0.858	0.081	86.06
SH XCS7-1002	BFDL	3.494	0.090	94.65
	REFDL	0.816	0.072	94.65
SH XCS7-1003	BFDL	3.914	0.085	97.73
	REFDL	0.801	0.071	97.73
Wustl	BFDL	3.002	0.095	94.26
	REFDL	0.816	0.076	94.26

The results in Table 6.3 are for the implemented BFDL method and its optimized counterpart REFDL against training procedures. It compared the training runtime, memory requirements and accuracy against model hidden layers (L) and virtual workers (VW) construct variations. As presented, the REFDL requires lower memory and time as tested with the SH XCST-1003 dataset. In addition, BFDL and REFDL federated models produce slightly better accuracy with four hidden layers (4L). As a result, the increment of hidden layers can influences effective federated learning in distributed settings. In each case, REFDL performs better than the BFDL across model architecture variations in terms of resource efficiency and faster on-device learning. This increase the chance of utilizing REFDL over BFDL in resource-constrained environments.

The illustration in Figures 6.6 and 6.7 present the memory and time savings of REFDL based on the reported results in Table 6.3. The results illustrate a better resource (memory and time) savings capability of REFDL against each training procedure. It demonstrates the significant memory savings of REFDL with 2VW-4L model architecture. With that, it can save 98.07% and 21.74% of training memory and

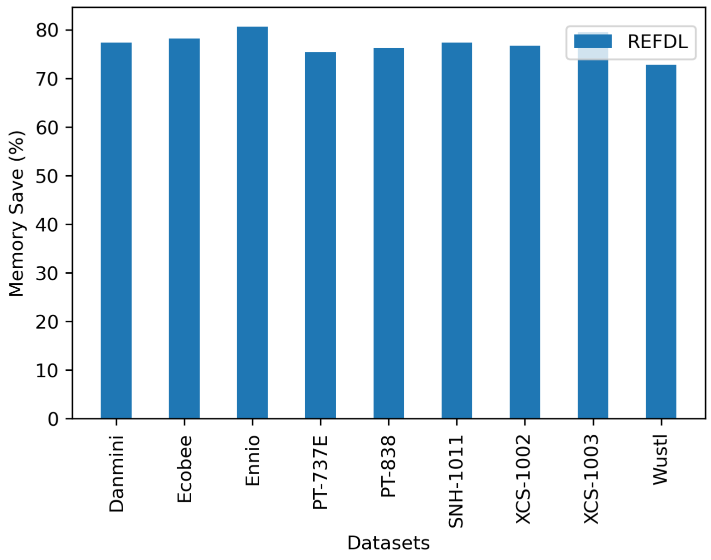

Figure 6.4: REFDL federated model training memory resources save against datasets (cumulative).

Table 6.3: Performance comparisons for FL training procedure on SH XCS7-1003 dataset (cumulative).

Procedure	Model	Memory MB	Time minutes	Test acc %
2VW-3L	BFDL	1.906	0.038	97.72
	REFDL	0.550	0.027	97.72
2VW-4L	BFDL	2.698	0.046	97.73
	REFDL	0.052	0.036	97.73
4VW-3L	BFDL	2.971	0.067	97.72
	REFDL	0.294	0.060	97.72
4VW-4L	BFDL	3.914	0.085	97.73
	REFDL	0.801	0.071	97.73

runtime using the 2VW-4L procedure without accuracy degradation. The results suggest that increments of virtual workers can facilitate better memory savings with three hidden layer network architecture. REFDL demonstrates more resource savings advantage even with a more complex model network architecture. This is important in a scenario in which a conventional shallow model cannot provide a better performance, in that scenario, a deeper model with more hidden layers can be used with our training procedure to benefit the resource efficiency and effective attack identification.

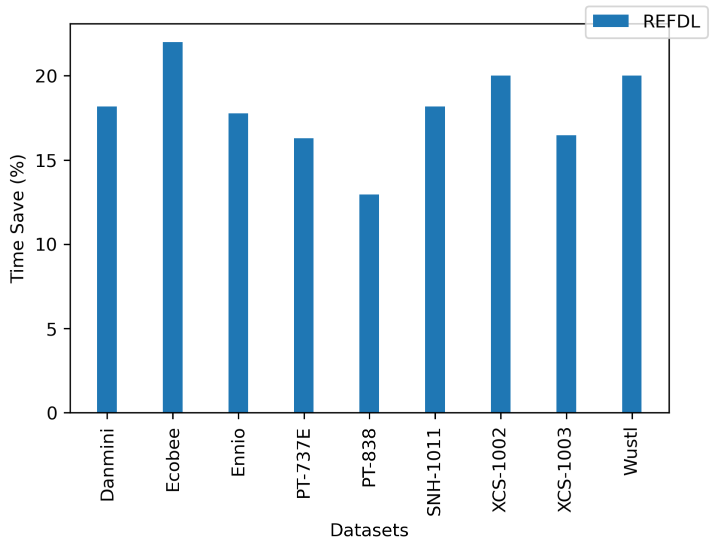

Figure 6.5: REFDL federated model training time resources save against datasets (cumulative).

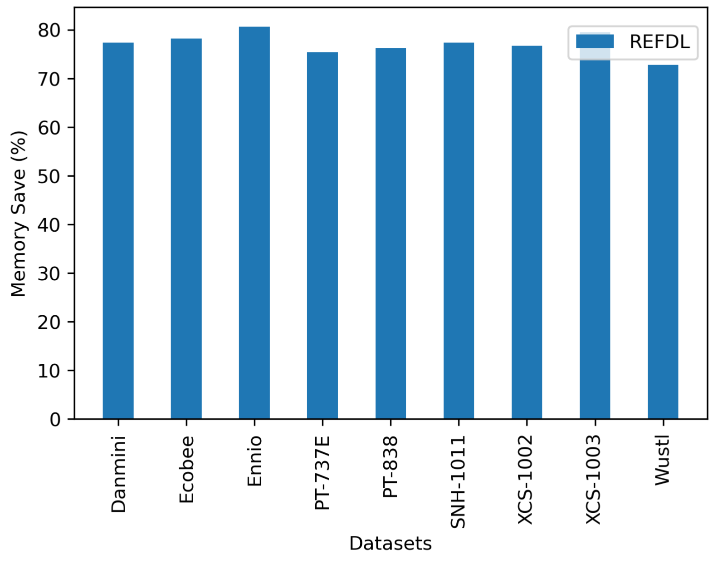

Figure 6.6: REFDL federated model training memory resources savings with XCS-1003 dataset.

To test the generalization and effectiveness of REFDL, In Table 6.4 we examined its performance over the MNIST image dataset with integrated CNN and FCNN using Algorithm 6 as the core model to train REFDL in the FL scenario. This is good to

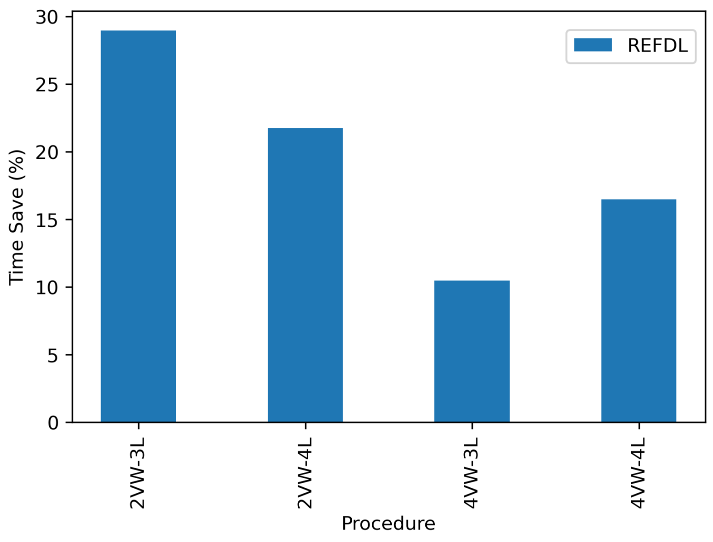

Figure 6.7: REFDL federated model training time resources savings with XCS-1003 dataset.

assess the method's performance over non-IoT datasets and examine its potential for on-device learning across different areas of research. In addition, this can enable us to exploit the resource-saving capability of CNN that offers a promise in image classification. We utilized the Pysyft WS simulated workers and examined the performance of the BFDL and REFDL techniques in federated training with the MNIST dataset. The REFDL demonstrates better detection accuracy than its BFDL counterparts with the CNN and FCNN model variation. As expected, it produces lower training execution time. The results show the benefit of regularization [188] and [179] on accuracy against DNN variation in FL setting with virtual client device workers. This attracts further investigation of REDFL in realistic settings with multiple edge-like-computing devices to further investigated its resource savings and effective detection capabilities.

Table 6.4: Simulated federated training performance comparison between BFDL and REFDL with MNIST dataset (cumulative).

Procedure	FL	Time minutes	Time save (%)	Test set acc %
FCNN-MNIST	BFDL	1.393	N/A	34.64
	REFDL	1.346	3.374	91.03
CNN-MNIST	BFDL	1.583	N/A	90.59
	REFDL	1.457	7.960	98.28

Low Precision Training

We investigated the effect of the proposed optimized training method in distributed FL, centralized and low precision schemes against the SH XCS-71003-WHT device. We present such results to examine the memory reduction with the utilized resource-efficient deep learning (REDL) model over its baseline deep learning (BDL) counterpart as tested in centralized and low precision settings (see Table 6.5). In each case, REDL demonstrates better memory savings. It can save 97.43%, 98.64% and 99.46% in centralized, low precision (FP16) and decentralized (FedAvg) settings. In addition to the significant memory reduction by the REDL model, across each training procedure, it outperforms the BDL model with reduced low precision 16-bit implementation that has become the de facto technique for increasing the energy efficiency of deep learning hardware [189]. As shown in Table 6.5, FP16 integration reduces the accuracy of the BDL model by 0.05 percentage points while reducing that of the REDL model by only 0.02 percentage points, respectively. This may be due to the automatic weight penalization of the REDL and regularization to ensure better convergence. In centralized and federated training procedures, both models demonstrate equal accuracy performance. The results show the significance of our optimized model compared with its benchmark counterpart. It further suggests that the proposed method is efficient and effective for on-device training in a distributed manner. Therefore, integrating FP16 with REDL can influence more memory reduction using the optimized training method. It demonstrated that FP16 integration does not influence REDL accuracy reduction in most cases. It can reduce the BDL classification accuracy across some datasets. Therefore, the regularized REDL can maintain a better accuracy with FP16 computations. Because of that, its federated method REFDL can be used in resource-constrained IoT environments for effective security monitoring.

Table 6.5: Performance comparisons against training procedure (cumulative).

Procedure	Model	Memory (MB)	Accuracy (%)
Centralized	BDL	3.969	97.72
	REDL	0.102	97.72
Low precision (FP16)	BDL	1.988	97.67
	REDL	0.027	97.70
Decentralized (FedAvg)	BFDL	10.15	97.72
	REFDL	0.055	97.72

Federated Model Performance Evaluation

Table 6.6 describes the federated model performance evaluated by test set accuracy, precision, recall and harmonic mean on randomly chosen datasets. As the chosen IoT DDoS and Wustl datasets are often imbalanced, the test accuracy alone cannot be used to assess model performance between the optimized and its baseline counterparts. For this reason, we utilized other metrics to assess the method's performance in proposing security solutions. These metrics are expressed in Equations 2.10, 2.11, 2.12 and 2.13, respectively. The optimized REFDL federated model maintains similar detection performance with the baseline across all metrics. The performance metrics result presented in Table 6.6 remained identical for models trained in centralized settings (BDL and REDL) against each dataset. In each case, accuracy, precision, recall and F1 score remained similar. The results indicate that the utilized number of virtual workers nodes in the federated settings had a minor influence on degrading model performance. This behaviour indicates the lightweight advantage and effectiveness of REFDL in detecting IoT attacks with similar F1-score performance as BFDL while saving significant resources.

Table 6.6: Testing performance comparisons across datasets.

Dataset	Model	Accuracy (%)	Precision	Recall	F1 score
IoT-DDoS	BFDL	83.34	0.8334	1.0000	0.9091
	REFDL	83.34	0.8334	1.0000	0.9091
Wustl	BFDL	94.26	0.9426	1.0000	0.9705
	REFDL	94.26	0.9426	1.0000	0.9705

6.4.2 Experimental Results (Testbed)

With Ennio Doorbell and Samsung SNH randomly selected IoT datasets, we investigated the memory consumption of training REFDL and BFDL across four GB-BXBT-2807 edge devices over wireless network testbed settings. The reported memory values in Table 6.7 are averaged based on the four devices. Since, we consider two different datasets, then we report the cumulative memory usage in MB. The demonstration results are based on the training performed based on 50 federated communication rounds and two local epochs iterations. The results show that the REFDL can detect IoT attacks with minimal memory than BFDL in real-time. The results prove the hypothesis stated in RQ2 by claiming that DNN can be trained to be resource efficient in realistic network settings for efficient IoT security monitoring without accuracy degradation.

We present averaged estimated real-time (cumulative) for training BFDL and REFDL based on the four GB-BXBT-2807 devices used in our testbed against the Ennio Doorbell and Samsung SNH IoT datasets in Figure 6.8. As our investigations considered realistic edge devices in FL settings, the execution time in the y-axis was reported in minutes. The REFDL requires less real-time than BFDL in training with each utilized dataset. The results demonstrate the effectiveness of REFDL in saving computational resources in resource-constrained environments over wireless network testbed settings.

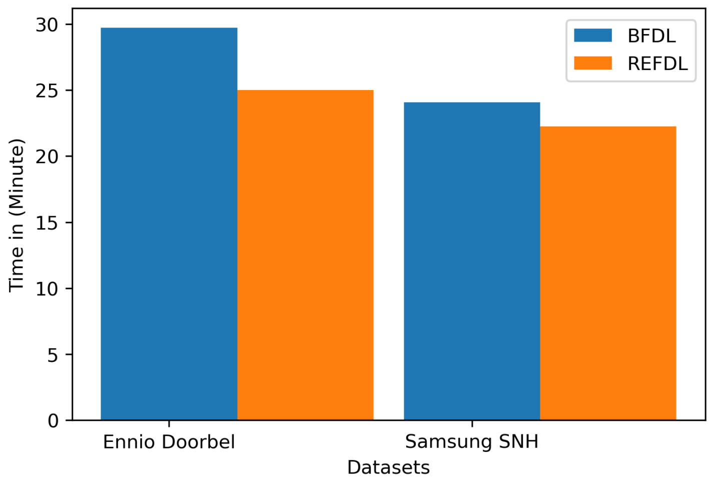

Figure 6.8: Federated model training time: REFDL vs BFDL using Ennio Doorbel and Samsung SNH datasets on real testbed (cumulative).

The illustration in Figure 6.9 shows the savings advantage in realistic settings over simulated counterparts. Even though they are two different procedures, this comparison is presented to assess the usability of our testbed and demonstrates its resource savings capability over a federated model developed using a personal desktop computer in a simulated setting. As expected, with realistic devices, the runtimes increase and even in that settings, REFDL demonstrates better savings than its baseline counterpart.

We examined the real-time savings of the REFDL over BFDL against the MNIST dataset. With the two local epoch iterations and 50 communication rounds, the times in Figure 6.10 show that REFDL is more efficient than BFDL across each training procedure. As expected, the MNIST-CNN federated training procedure is more computationally expensive than the MNIST-FCNN. In that context, the FCNN DNN variant of REFDL can be an appropriate choice for on-device learning if savings resources are the target objectives. In addition, the MNSIT-CNN that utilized our optimized training

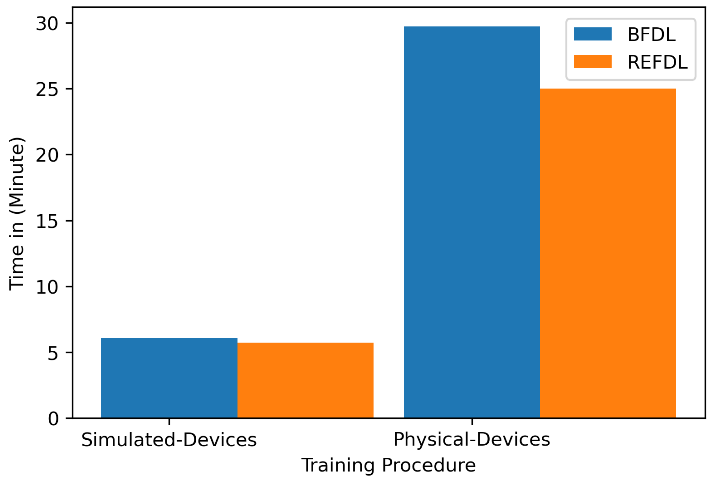

Figure 6.9: Federated model training execution time: REFDL vs BFDL in simulation and realistic testbed settings using Ennio Doorbell dataset. (cumulative)

Table 6.7: Federated model training memory: REFDL vs BFDL using Ennio Doorbell and Samsung SNH datasets on real testbed (cumulative).

Dataset	Model	Memory MB	Memory save %	Test acc %
Ennio Doorbell	BFDL	33.965	N/A	89.00
	REFDL	31.981	5.84	89.00
Samsung SNH	BFDL	32.519	N/A	86.10
	REFDL	30.550	6.05	86.10

procedure can save resources better than its baseline counterparts. As such, REFDL stands more suitable method for deployment in an IoT resource environment.

The illustration in Figure 6.11 shows the convergence accuracy using 2 local epoch iterations and 50 federated communication rounds of REFDL and BFDL against DNN variants with the MNIST dataset. In each training procedure with the two DNN variants, REFDL stands to be a better model than BFDL. It can classify image samples accurately with integrated CNN and FCNN (DNN) model variants. The result suggests the advantage of optimization mechanisms in producing a global deep federated model. It further demonstrates the effectiveness of integrating CNN in the FL method to improve accuracy performance. This is good as it leverages the tradeoff between each DNN

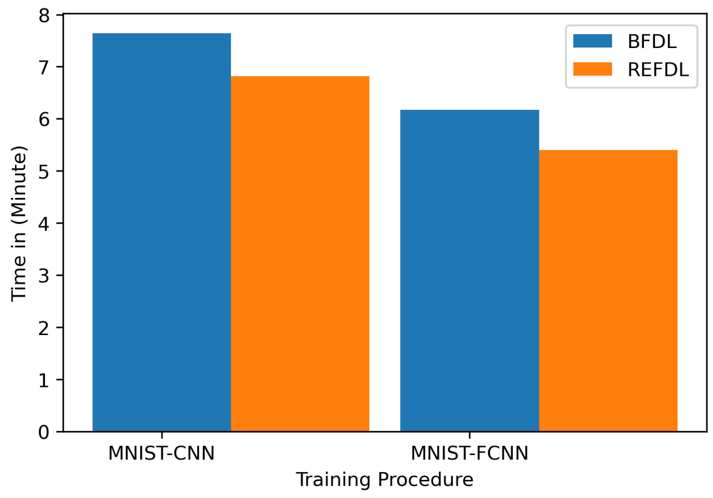

Figure 6.10: Federated model training time: REFDL vs BFDL using MNIST dataset on real testbed (cumulative).

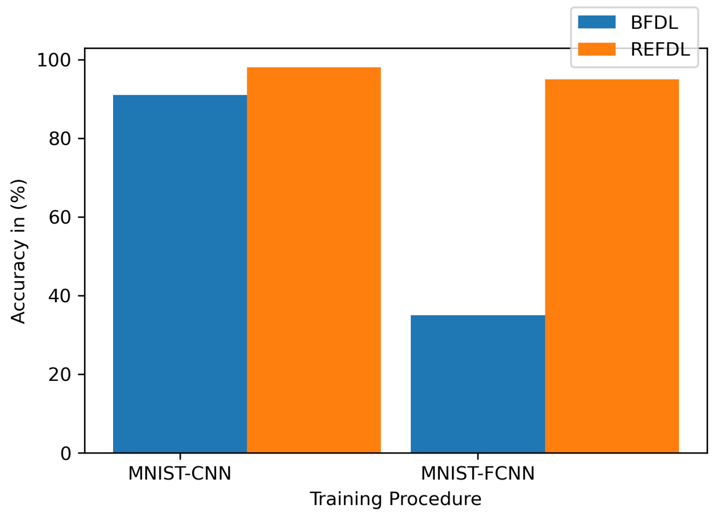

Figure 6.11: Federated model accuracy comparison between REFDL and BFDL with MNIST dataset.

model during on-device learning.

To test the effectiveness and faster learning of REFDL on GB-BXBT-2807 testbed federated settings, we vary the epoch iterations from (1 - 5, see Figure 6.12) using the FCNN-MNIST procedure. For a fair comparison, in each training procedure, the federated communication round is 50. In that aspect, we can assess the performance of

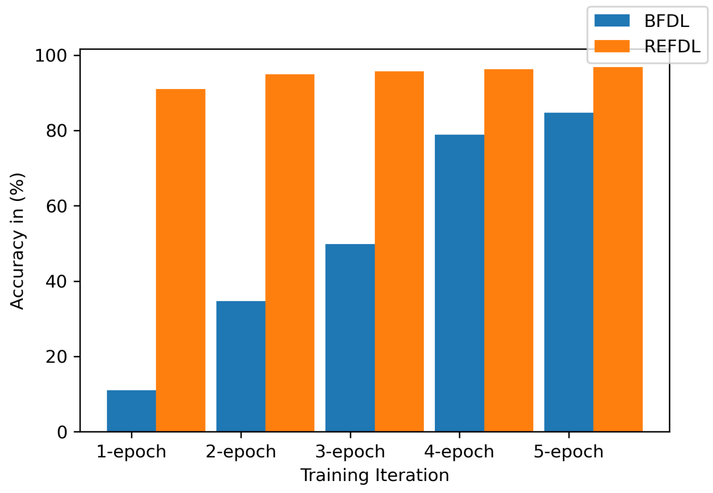

Figure 6.12: Federated model accuracy performance with epochs using 50 communication rounds: REFDL vs BFDL using FCNN-MNIST procedure.

each federated method in real time. As shown in Figure 6.12, the optimized REFDL can achieve a better accuracy even with one local epoch iteration and 50 federated communication rounds. This trends of providing higher accuracy remain stable across each local epoch iterations and the stated communication rounds. The result demonstrates REFDL appropriateness and faster learning capability across edge devices, especially with the integrated FCNN model. REFDL minimum number of epoch requirements is advantageous, especially in an environmental setting such as IoT with inherited limited memory resources.

In Table 6.8, we present the performance comparison between REFDL and BFDL using the federated training procedure CNN-MNIST over different communication rounds. The reported results are for using one local epoch iteration. In each communication round, REFDL demonstrated better accuracy than its baseline counterparts. Using the same experimental setting but with two local epoch iterations, REFDL and BFDL can accurately classify the MNIST dataset with 99% and 93%, respectively, in which REFDL provides better accuracy. The results suggest the capability of REFDL in classifying both IoT and non-IoT datasets in real time with better accuracy.

Table 6.8: Federated model accuracy: REFDL vs BFDL against CNN-MNIST training procedure.

Federated rounds	Model	Test acc %
50	BFDL	89.00
	REFDL	97.00
100	BFDL	89.00
	REFDL	97.00

6.4.3 Summary

FL is a distributed ML with support for on-device learning in decentralized edge devices over a network. This scheme enables data privacy across multiple clients. In this chapter, we investigated the feasibility of running FL training in decentralized resource-constrained environments with IoT datasets. This can be an essential step in providing security solutions for IoT devices. For this purpose, we utilized FedAvg (BFDL algorithm) with model optimization techniques proposed in chapter 4 to produce REFDL that is more resource-efficient with accurate attack detection. This is to investigate the possibility of reducing memory consumption during federated model training with DNN, intending to use it as a security solution in resource-constrained environments. By utilizing the FCNN and CNN models, we proposed a resource-efficient REFDL for the effective detection of cyber attacks on IoT devices. The effectiveness of REFDL was tested using various IoT and non-IoT benchmark datasets in simulated FL settings with various virtual client workers. Experimental results showed that the proposed REFDL outperform its benchmark counterparts for memory efficiency and accuracy performance. This is due to the utilization of the optimized training procedure of the clients model in the federated training, thus the cumulative savings are higher with the optimized REFDL than its baseline counterparts. In addition, the aggregation of models in federated training to build REFDL can influence faster learning capability compared with the baseline training. However, these initial experimental results are encouraging and warrant further investigation of the resource savings capability of the utilized computational nodes in a realistic federated environment. Therefore, we investigate the deployment capability of the REFDL in a real wireless network setting with physically connected devices and examine its capabilities to detect IoT attacks in near real-time in an FL setting. Extensive experiments with eight IoT datasets and one image dataset in simulated environments demonstrate the resource savings capability of REFDL over BFDL. While examining the resource efficiency of REFDL using GB-BXBT-2807-edge computing-like devices in realistic testbed settings, it demonstrates effectiveness, low complexity and efficiency than its counterparts. It can detect IoT attacks accurately using minimal resources better than its counterparts in real-time. In addition, REFDL requires fewer epochs to produce a more accurate FL model than its counterparts. For instance, using one local epoch iteration and 50 communication rounds, it can attain an accuracy of 91%, which is 80% better than that produced by its counterparts under the same federated settings.

Chapter 7

Conclusion

The IoT is an advanced technology serving as an ecosystem used in smart homes, smart cities and many automation systems. It uses multiple standard protocols in the forms of Bluetooth and WiFi to connect various devices in a network. IoT devices are task-specific and AI technologies can be integrated to collect and exchange data. The collected data are essential in conducting many applications in various domains, particularly hospitals and manufacturing industries. In addition, prominent industries utilized IoT devices conveniently to ease their daily activities. However, with the rise of the cyber attacks and botnet threats, IoT devices are the potential targets for various cyber attacks in the security landscape. In addition, the resource-constrained nature of IoT devices limits the direct deployment of AI techniques that can be used to monitor, assess and detects IoT cyber attacks. Therefore, this thesis investigated the feasibility of using effective, efficient and robust AI techniques to address IoT security challenging issues. For proof of concept, we investigated the research questions listed in chapter 1.

Based on those research questions, this thesis systematically focused on developing efficient, robust and effective IoT security solutions as described in chapters 3, 4, 5 and 6. This enables us to investigate the procedure of proofing the stated hypothesis. As such, we present the following contributions:

1. In chapter 3, we investigate several training strategies for reducing the computational complexity of using ML methods for IoT cyber security monitoring. Initial experimentation investigated the feasibility of using the GI and PCA feature reduction techniques to investigate their capabilities of saving computational resources

for IoT cyber security monitoring. To this end, we examined the integration of feature reduction techniques to compare how GI and PCA reduce the computational expense of running the AIS algorithm. Our findings suggest that the PCA integration can reduce memory and time consumption better than the GI counterparts. A disadvantage of the PCA integration is the inability to increase the detection accuracy of the AIS algorithm across the few tested IoT cyber security datasets. Following the motivational results of the feature reduction techniques and the limitations of their detection capabilities, we introduced an optimized training procedure for LGBM (DTEM) to reduce its computational complexity and resource consumption nature in IoT security monitoring. This served as a novel training procedure for LGBM (DTEM) that leverages previous researches in cyber security monitoring and IoT intrusion detection. This training procedure was capable of incorporating memory and time constraints while optimizing the LGBM model. Experiments on publicly available benchmark IoT datasets demonstrate that the proposed procedure reduces required computational resources for training and testing of the LGBM method without significantly degrading the detection capabilities. This is due to the factoring out of the less computational expensive parameters of the LGBM using the optimized training method. Overall, it outperforms each employed DTEM counterpart in our experiments. Our findings suggest that LGBM can be optimized using the proposed training procedure. These motivational results demonstrate the feasibility of improving the training strategies of conventional ML techniques to reduce their computational complexity in the context of IoT security monitoring.

2. In chapter 4, we explore the resource efficiency and effective training strategies for the FCNN DNN-based method to develop appropriate IoT security solutions. We are more concerned with developing more promising AI security solutions that can outperform conventional ML algorithms in terms of significant resource savings, better learning capabilities and effective attack detection. For this purpose, we introduced a novel training procedure for the DNN method that outperforms most conventional ML algorithms in cyber security monitoring. In our proposed DNN optimization training procedure, we in-cooperated memory and time constraints as a threshold to find the best trade-off between reducing model complexity and maintaining an appropriate performance. To this end, an FCNN model is used to assess our optimized training procedure aiming to create effective and efficient

DNN security solutions suitable for resource constrained IoT environments. We utilized an appropriate regularization method with simulated micro-batching to reduce the computational complexity of the training procedure of FCNN model to build resource-efficient REDNN-based IoT security solutions. Results are promising as the resulting REDNN remains efficient and effective compared with its benchmark counterparts in various evaluation frameworks across many commercially tailored IoT device datasets. Our finding suggests that FCNN can offer better and more efficient security solutions in IoT environments than its counterparts and other ML methods. This is due to the capability of DNN to learn the complex patterns present in a dataset.

3. In chapter 5, we further explore our optimized training method for DNN in a robust scenario to investigate its potentiality to resist various adversarial attacks generated in IoT network environments. We are more concerned with developing a robust DNN technique for IoT security monitoring in the presence of adversarially generated attacks. To this end, we investigated the resilience of REDNN against various perturbation methods. This is to leverage our proposed REDNN method to detect adversarial samples created using IoT commercial and device-centric datasets. Experimental results suggest that REDNN can resist to adversarial perturbations better than the employed AI methods.

4. Finally, in chapter 6, we investigate our proposed REDNN in the FL scenario using simulated and realistic network settings. We start by exploring on-device learning with virtual edge devices and illustrate the development of a network testbed for training DNN in a decentralized edge computing environment using the FL scheme to build appropriate IoT security solutions. The main goal is to design a novel and realistic testbed to evaluate the effectiveness of REDNN. This is good to leverage REDNN feasibility as a method that can efficiently detect attacks within various IoT devices and preserve their data against third parties. We utilized a Federated Averaging (FedAvg) to assess the REDNN performance in realistic IoT network settings rather than in a simulation environment. The federated integration can preserve the training data while creating a better global model in the security context. Results from simulation and network testbed demonstrate the effectiveness of REDNN in creating a resource-efficient federated model. The results suggest that the DNN can be optimized and trained in a federated scenario to grant data privacy, saves computational resources and detect attacks. This is good, especially in an environment with limited computational resources (IoT) containing various

connected decentralized edge devices. The REDNN can be feasibly deployed as a framework in a resource-constrained environment to serve as a security solution.

From the findings demonstrated in this thesis, there is a benefit for an effective, efficient, and robust security mechanism to detect attacks in IoT resource-constrained network environments. In particular, for DNN, we have demonstrated that exploration of regularization, and simulated micro-batching techniques can enable the development of training procedures that can outperform the baseline methods. Investigation results suggest that the resulting model can serve as a novel security scheme in an IoT network environment. As a result, we have demonstrated the deployment capability of DNN trained using our proposed training method in a realistic network with decentralized edge devices. In this scenario, DNN can use a federated strategy to preserve the utilized training data while developing a global and efficient security mechanism. This is good for on-device learning, especially in a resource-constrained environment that focuses on the security of the connected devices. Therefore, the descriptions of findings from this thesis support our hypothesis.

7.1 Future Work

In this thesis, we leveraged ML and DNN to develop IoT security monitoring solutions. With the DNN, we proposed a technique that can detect IoT attacks in an efficient, effective and robust scenario. Our proposed DNN technique achieved promising results in terms of resource efficiency, accurate attack detection and resistance to adversarial perturbations. However, we have identified certain limitations regarding our study and outlined possible future research directions.

Looking at the training strategy described in chapter 3, we adhere that the results we obtained regarding the feature reduction on fewer publicly benchmark cyber security datasets demonstrate that PCA is a better candidate to reduce the computational complexity of using the AIS algorithm for IoT cyber security. However, only one device of the N-BaIoT dataset are considered in our experiments, it would be interesting to test various device centric datasets and compare how feature reduction can performs in terms of resource efficiency. Regarding the LGBM exploitation, the results we obtained using the optimized training procedure for LGBM are good evidence that a lightweight ML method can be promising for IoT security monitoring. An essential investigation from chapter 3, is the balance-off between efficient detection and memory and time consumption reduction at the training and testing stage. However, it would be interesting to investigate our algorithm performance against multiple benchmark datasets from various domains with other conventional ML techniques and examine the resource consumption and effective attack detection capabilities.

In chapter 4, we leveraged DNN for IoT security monitoring in the context of resource efficiency and effective attack detection in IoT network environments. With the exploration of regularization and simulated micro-batching imposed by memory and execution time constraints, we demonstrated that DNN can be trained in an optimized way in a resource-constrained environment to save more resources and demonstrate an efficient attack detection better than conventional ML algorithms. However, further research can be considered to exploit REDNN learning capabilities with non-IoT datasets to benefit from the training and testing resource-saving advantage of the optimized algorithm. This can be useful for testing the generalization ability of REDNN in saving computational resources in running DNN models across various fields.

In chapter 5, we explore the potentiality of our DNN training method in providing robust IoT security solutions. In this aspect, we realized the capability of REDNN in detecting adversarial attacks crafted from device-centric IoT datasets is better than the baseline and other state-of-the-art ML techniques. However, in our experiments we consider the most commonly perturbations methods used in crafting the adversarial samples. As IoT device are dynamic in nature, in future, we plan to extend the DNN capability to resist other perturbations techniques that are not utilized in this thesis and captured from complex IoT and non-IoT datasets. REDNN's resilience to privacy leakage and model inference attacks would also be interesting to study.

Finally, we are passionate about further exploring DNN detection capabilities in realistic network settings using multiple devices. In chapter 6, we have seen DNN's potentiality in reducing computational resources while detecting attacks in virtual and realistic like IoT network environments. In particular, in a federated environment that supports on-device learning. These results are encouraging and attract further investigation for utilizing more client devices in the network, particularly, over wired and wireless settings. Since the aim of this study is to investigate the ability to reduce resource consumption while maintaining the same (higher) accuracy as REDNN's counterparts, we did not directly attack client devices using offensive tools (e.g. Kali Linux) in our experiments. Instead, we replayed benchmark datasets on client devices and fed them into our algorithms to assess their real-time detection capabilities. However, in the future, it would be interesting to attack client devices using offensive tools to investigate the detection capabilities of such attacks by our algorithm. In addition, investigating the resilient capability of the REFDL to enhance its security robustness against adversarial attacks in a realistic network setting with various connected edge devices other than the ones considered in this thesis would be interesting. This will enable examining the resource efficiency and security monitoring performance of our proposed method across multiple decentralized edge devices.

Bibliography

[1] Bojan J. Internet of Things statistics for 2022 - Taking Things Apart; 2022. Available from: https://dataprot.net/statistics/iot-statistics/.

[2] Jenalea H. Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says; 2017. Available from: https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says.

[3] Dong B, Shi Q, Yang Y, Wen F, Zhang Z, Lee C. Technology evolution from self-powered sensors to AIoT enabled smart homes. Nano Energy. 2021;79:105414.

[4] Liu S. Iot market size worldwide 2017-2025. URL: https://www statista com/statistics/976313/global-iot-market-size. 2020.

[5] Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, et al. Understanding the mirai botnet. In: 26th {USENIX} security symposium ({USENIX} Security 17); 2017. p. 1093-110.

[6] Alexa H. Cyber Attacks on IoT Devices Are Growing at Alarming Rates [Encryption Digest 64]; 2021. Available from: https://www.venafi.com/blog/cyber-attacks-iot-devices-are-growing-alarming-rates-encryption-digest-64.

[7] Zandberg K, Schleiser K, Acosta F, Tschofenig H, Baccelli E. Secure firmware updates for constrained iot devices using open standards: A reality check. IEEE Access. 2019;7:71907-20.

[8] Merenda M, Porcaro C, Iero D. Edge machine learning for ai-enabled iot devices: A review. Sensors. 2020;20(9):2533.

[9] Zhang H, Li JL, Liu XM, Dong C. Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection. Future Generation Computer Systems. 2021;122:130-43.

[10] Moustafa N, Hu J, Slay J. A holistic review of Network Anomaly Detection Systems: A comprehensive survey. Journal of Network and Computer Applications. 2019;128:33-55.

[11] HB BG, Poornachandran P, KP S, et al. Deep-Net: Deep neural network for cyber security use cases. arXiv preprint arXiv:181203519. 2018.

[12] Aggarwal CC, et al. Neural networks and deep learning. Springer. 2018;10:978-3.

[13] Tuna OF, Catak FO, Eskil MT. Exploiting epistemic uncertainty of the deep learning models to generate adversarial samples. Multimedia Tools and Applications. 2022:1-22.

[14] Lasisi A, Ghazali R, Herawan T. Application of real-valued negative selection algorithm to improve medical diagnosis. In: Applied Computing in Medicine and Health. Elsevier; 2016. p. 231-43.

[15] Csizmadia G, Liszkai-Peres K, Ferdinandy B, Miklósi Á, Konok V. Human activity recognition of children with wearable devices using LightGBM machine learning. Scientific Reports. 2022;12(1):1-10.

[16] Holbrook L, Alamaniotis M. A good defense is a strong DNN: defending the IoT with deep neural networks. In: Machine Learning Paradigms. Springer; 2020. p. 125-45.

[17] Mahmud R, Kotagiri R, Buyya R. Fog computing: A taxonomy, survey and future directions. In: Internet of everything. Springer; 2018. p. 103-30.

[18] Arivazhagan C, Natarajan V. A Survey on Fog computing paradigms, Challenges and Opportunities in IoT. In: 2020 International Conference on Communication and Signal Processing (ICCSP). IEEE; 2020. p. 0385-9.

[19] Zevala C. IoT Memory: An Overview of the Options. Accessed: Sep; 2018.

[20] Atmel. SMART ARM-Based Wireless Microcontroller; 2016. Available from: http://ww1.microchip.com/downloads/en/devicedoc/sam-r21_datasheet.pdf.

[21] Bormann C, Ersue M, Keranen A. Terminology for constrained-node networks; 2014.

[22] Paloalto I. Impacts of Cyberattacks on IoT Devices. Accessed: Jan; 2020.

[23] Baig ZA, Sanguanpong S, Firdous SN, Nguyen TG, So-In C, et al. Averaged dependence estimators for DoS attack detection in IoT networks. Future Generation Computer Systems. 2020;102:198-209.

[24] Goodin D. BrickerBot, the permanent denial-of-service botnet, is back with a vengeance. Ars Technica. 2017.

[25] Abomhara M, Køien GM. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. Journal of Cyber Security and Mobility. 2015:65-88.

[26] Mohammadnia H, Slimane SB. IoT-NETZ: Practical spoofing attack mitigation approach in SDWN network. In: 2020 Seventh International Conference on Software Defined Systems (SDS). IEEE; 2020. p. 5-13.

[27] Rajan A, Jithish J, Sankaran S. Sybil attack in IOT: Modelling and defenses. In: 2017 international conference on advances in computing, communications and informatics (ICACCI). IEEE; 2017. p. 2323-7.

[28] Hachemi FE, Mana M, Bensaber BA. Study of the impact of sinkhole attack in iot using shewhart control charts. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE; 2020. p. 1-5.

[29] Jedh M, Othmane LB, Ahmed N, Bhargava B. Detection of message injection attacks onto the can bus using similarities of successive messages-sequence graphs. IEEE Transactions on Information Forensics and Security. 2021;16:4133-46.

[30] Stasinopoulos A, Ntantogian C, Xenakis C. Commix: Automating evaluation and exploitation of command injection vulnerabilities in web applications. International Journal of Information Security. 2019;18(1):49-72.

[31] Torres-Arias S, Afzali H, Kuppusamy TK, Curtmola R, Cappos J. in-toto: Providing farm-to-table guarantees for bits and bytes. In: 28th USENIX Security Symposium (USENIX Security 19); 2019. p. 1393-410.

[32] Rahim R, Nurdiyanto H, Abdullah D, Hartama D, Napitupulu D, et al. Keylogger application to monitoring users activity with exact string matching algorithm. In: Journal of Physics: Conference Series. vol. 954. IOP Publishing; 2018. p. 012008.

[33] D'Orazio CJ, Choo KKR, Yang LT. Data exfiltration from Internet of Things devices: iOS devices as case studies. IEEE Internet of Things Journal. 2016;4(2):524-35.

[34] Sontiq I. Identity Theft Resource Cener's 2021 Annual Data Breach Report Sets New Record for Number of Compromises. Accessed: Jan; 2022.

[35] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. nature. 2015;518(7540):529-33.

[36] Altman N, Krzywinski M. Ensemble methods: bagging and random forests. Nature Methods. 2017;14(10):933-5.

[37] Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems. 2018;31.

[38] Cutler A, Cutler DR, Stevens JR. Random forests. In: Ensemble machine learning. Springer; 2012. p. 157-75.

[39] Wager S, Athey S. Estimation and inference of heterogeneous treatment effects using random forests. Journal of the American Statistical Association. 2018;113(523):1228-42.

[40] Schapire RE. Explaining adaboost. In: Empirical inference. Springer; 2013. p. 37-52.

[41] Li P. Robust logitboost and adaptive base class (abc) logitboost. arXiv preprint arXiv:12033491. 2012.

[42] Pandey P, Prabhakar R. An analysis of machine learning techniques (J48 & AdaBoost)-for classification. In: 2016 1st India International Conference on Information Processing (IICIP). IEEE; 2016. p. 1-6.

[43] Farris FA. The Gini index and measures of inequality. The American Mathematical Monthly. 2010;117(10):851-64.

[44] Liu H, Zhou M, Lu XS, Yao C. Weighted Gini index feature selection method for imbalanced data. In: IEEE ICNSC; 2018. p. 1-6.

[45] D'Ambrosio A, Tutore VA. Conditional classification trees by weighting the Gini impurity measure. In: New perspectives in statistical modeling and data analysis. Springer; 2011. p. 273-80.

[46] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in neural information processing systems; 2017. p. 3146-54.

[47] Jiang J, Cui B, Zhang C, Fu F. Dimboost: Boosting gradient boosting decision tree to higher dimensions. In: Proceedings of the 2018 International Conference on Management of Data; 2018. p. 1363-76.

[48] Corporation M. LGBM Tuning Parameters; 2022. Available from: `https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html`.

[49] Nematzadeh S, Kiani F, Torkamanian-Afshar M, Aydin N. Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases. Computational Biology and Chemistry. 2022;97:107619.

[50] Suthaharan S. Support vector machine. In: Machine learning models and algorithms for big data classification. Springer; 2016. p. 207-35.

[51] Li Z, Liu F, Yang W, Peng S, Zhou J. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Transactions on Neural Networks and Learning Systems. 2021.

[52] Hosseinzadeh M, Rahmani AM, Vo B, Bidaki M, Masdari M, Zangakani M. Improving security using SVM-based anomaly detection: issues and challenges. Soft Computing. 2021;25(4):3195-223.

[53] Zhu F, Chen W, Yang H, Li T, Yang T, Zhang F. A quick negative selection algorithm for one-class classification in big data era. Mathematical Problems in Engineering. 2017;2017.

[54] Pamukov ME, Poulkov VK. Multiple negative selection algorithm: Improving detection error rates in IoT intrusion detection systems. In: IEEE IDAACS. vol. 1; 2017. p. 543-7.

[55] Rashid N, Iqbal J, Mahmood F, Abid A, Khan US, Tiwana MI. Artificial immune system–negative selection classification algorithm (NSCA) for four class electroencephalogram (EEG) signals. Frontiers in human neuroscience. 2018;12:439.

[56] Hoang DH, Nguyen HD. A PCA-based method for IoT network traffic anomaly detection. In: IEEE 20th ICACT; 2018. p. 381-6.

[57] Nielsen MA. Neural networks and deep learning. vol. 25. Determination press San Francisco, CA, USA; 2015.

[58] Li CL, Sohn K, Yoon J, Pfister T. Cutpaste: Self-supervised learning for anomaly detection and localization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2021. p. 9664-74.

[59] Goh J, Adepu S, Tan M, Lee ZS. Anomaly detection in cyber physical systems using recurrent neural networks. In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE). IEEE; 2017. p. 140-5.

[60] Liang T, Glossner J, Wang L, Shi S, Zhang X. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing. 2021;461:370-403.

[61] Mellempudi N, Kundu A, Mudigere D, Das D, Kaul B, Dubey P. Ternary neural networks with fine-grained quantization. arXiv preprint arXiv:170501462. 2017.

[62] Chauvin Y, Rumelhart DE. Backpropagation: theory, architectures, and applications. Psychology press; 2013.

[63] Shakya A, Biswas M, Pal M. Parametric study of convolutional neural network based remote sensing image classification. International Journal of Remote Sensing. 2021;42(7):2663-85.

[64] Al Tobi AM, Duncan I. KDD 1999 generation faults: A review and analysis. Journal of Cyber Security Technology. 2018;2(3-4):164-200.

[65] DARPA Intrusion Detection Dataset;. Available from: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-data-set.

[66] KDD Cup 1999;. Available from: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[67] Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 military communications and information systems conference (MilCIS). IEEE; 2015. p. 1-6.

[68] The UNSW-NB15 Dataset;. Available from: https://research.unsw.edu.au/projects/unsw-nb15-dataset.

[69] Ahmad M, Riaz Q, Zeeshan M, Tahir H, Haider SA, Khan MS. Intrusion detection in internet of things using supervised machine learning based on application and transport layer features using UNSW-NB15 data-set. EURASIP Journal on Wireless Communications and Networking. 2021;2021(1):1-23.

[70] Meidan Y, Bohadana M, Mathov Y, Mirsky Y, Shabtai A, Breitenbacher D, et al. N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing. 2018;17(3):12-22.

[71] N-BaIoT;. Available from: https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT.

[72] Koroniotis N, Moustafa N, Sitnikova E, Turnbull B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. Future Generation Computer Systems. 2019;100:779-96.

[73] The BoT-IoT Dataset;. Available from: https://research.unsw.edu.au/projects/bot-iot-dataset.

[74] Mirsky Y, Doitshman T, Elovici Y, Shabtai A. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection; 2018.

[75] Ktsune Network Attack Dataset;. Available from: https://archive.ics.uci.edu/ml/datasets/Kitsune+Network+Attack+Dataset.

[76] Teixeira MA, Salman T, Zolanvari M, Jain R, Meskin N, Samaka M. SCADA system testbed for cybersecurity research using machine learning approach. Future Internet. 2018;10(8):76.

[77] WUSTL-IIOT-2021 Dataset for IIoT Cybersecurity Research;. Available from: WUSTL-IIOT-2021DatasetforIIoTCybersecurityResearch.

[78] Siddharth M. IoT-DDoS dataset; 2020. Available from: https://www.kaggle.com/siddharthm1698/ddos-botnet-attack-on-iot-devices.

[79] Sánchez PMS, Valero JMJ, Celdrán AH, Bovet G, Pérez MG, Pérez GM. A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets. IEEE Communications Surveys Tutorials. 2021;23(2):1048-77.

[80] Elrawy MF, Awad AI, Hamed HF. Intrusion detection systems for IoT-based smart environments: a survey. Journal of Cloud Computing. 2018;7(1):1-20.

[81] Bhunia SS, Gurusamy M. Dynamic attack detection and mitigation in IoT using SDN. In: 2017 27th International telecommunication networks and applications conference (ITNAC). IEEE; 2017. p. 1-6.

[82] Nskh P, Varma MN, Naik RR. Principle component analysis based intrusion detection system using support vector machine. In: IEEE RTEICT; 2016. p. 1344-50.

[83] Lopez-Martin M, Carro B, Sanchez-Esguevillas A. IoT type-of-traffic forecasting method based on gradient boosting neural networks. Future Generation Computer Systems. 2020;105:331-45.

[84] Moustafa N, Turnbull B, Choo KKR. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. IEEE Internet of Things Journal. 2018;6(3):4815-30.

[85] Tang D, Tang L, Dai R, Chen J, Li X, Rodrigues JJ. MF-Adaboost: LDoS attack detection based on multi-features and improved Adaboost. Future Generation Computer Systems. 2020;106:347-59.

[86] Resende PAA, Drummond AC. A survey of random forest based methods for intrusion detection systems. ACM Computing Surveys (CSUR). 2018;51(3):1-36.

[87] Hasan MAM, Nasser M, Pal B, Ahmad S. Support vector machine and random forest modeling for intrusion detection system (IDS). Journal of Intelligent Learning Systems and Applications. 2014;2014.

[88] Farnaaz N, Jabbar M. Random forest modeling for network intrusion detection system. Procedia Computer Science. 2016;89:213-7.

[89] Ikram ST, Cherukuri AK. Improving accuracy of intrusion detection model using PCA and optimized SVM. Journal of computing and information technology. 2016;24(2):133-48.

[90] Haripriya P, Anju J. An AIS based anomaly detection system. In: 2017 International Conference on Computing Methodologies and Communication (ICCMC). IEEE; 2017. p. 708-11.

[91] Latif S, Idrees Z, Zou Z, Ahmad J. DRaNN: A deep random neural network model for intrusion detection in industrial IoT. In: 2020 International Conference on UK-China Emerging Technologies (UCET). IEEE; 2020. p. 1-4.

[92] Shareena J, Ramdas A, AP H, et al. Intrusion detection system for iot botnet attacks using deep learning. SN Computer Science. 2021;2(3):1-8.

[93] Kumar V, Kumar A, Garg S, Payyavula S. Boosting Algorithms to Identify Distributed Denial-of-Service Attacks. In: Journal of Physics: Conference Series. vol. 2312. IOP Publishing; 2022. p. 012082.

[94] Sharafaldin I, Lashkari AH, Hakak S, Ghorbani AA. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In: 2019 International Carnahan Conference on Security Technology (ICCST). IEEE; 2019. p. 1-8.

[95] Okey OD, Maidin SS, Adasme P, Lopes Rosa R, Saadi M, Carrillo Melgarejo D, et al. BoostedEnML: Efficient Technique for Detecting Cyberattacks in IoT Systems Using Boosted Ensemble Machine Learning. Sensors. 2022;22(19):7409.

[96] Cai W, Wei R, Xu L, Ding X. A method for modelling greenhouse temperature using gradient boost decision tree. Information Processing In Agriculture. 2021.

[97] Bileschi ML, Belanger D, Bryant DH, Sanderson T, Carter B, Sculley D, et al. Using deep learning to annotate the protein universe. Nature Biotechnology. 2022:1-6.

[98] Zhang Y, Krishnan V, Pi J, Kaur K, Srivastava A, Hahn A, et al. Cyber physical security analytics for transactive energy systems. IEEE Transactions on Smart Grid. 2019;11(2):931-41.

[99] Fadlullah ZM, Tang F, Mao B, Kato N, Akashi O, Inoue T, et al. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. IEEE Communications Surveys & Tutorials. 2017;19(4):2432-55.

[100] Chen C, Liu Z, Wan S, Luan J, Pei Q. Traffic flow prediction based on deep learning in internet of vehicles. IEEE transactions on intelligent transportation systems. 2020;22(6):3776-89.

[101] Abdellah AR, Koucheryavy A. Deep learning with long short-term memory for iot traffic prediction. In: Internet of Things, Smart Spaces, and Next Generation Networks and Systems. Springer; 2020. p. 267-80.

[102] Li X, Liu H, Wang W, Zheng Y, Lv H, Lv Z. Big data analysis of the internet of things in the digital twins of smart city based on deep learning. Future Generation Computer Systems. 2022;128:167-77.

[103] Jung W, Zhao H, Sun M, Zhou G. IoT botnet detection via power consumption modeling. Smart Health. 2020;15:100103.

[104] Pour MS, Mangino A, Friday K, Rathbun M, Bou-Harb E, Iqbal F, et al. On data-driven curation, learning, and analysis for inferring evolving internet-of-Things (IoT) botnets in the wild. Computers & Security. 2020;91:101707.

[105] Tang J, Sun D, Liu S, Gaudiot JL. Enabling deep learning on IoT devices. Computer. 2017;50(10):92-6.

[106] Iandola F, Keutzer K. Keynote: small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures. In: 2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS). IEEE; 2017. p. 1-10.

[107] Shi Z, Xie X, Lu H, Yang H, Kadoch M, Cheriet M. Deep-Reinforcement-Learning-Based Spectrum Resource Management for Industrial Internet of Things. IEEE Internet of Things Journal. 2020;8(5):3476-89.

[108] Shen S, Li R, Zhao Z, Liu Q, Liang J, Zhang H. Efficient Deep Structure Learning for Resource-Limited IoT Devices. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE; 2020. p. 1-6.

[109] Lawrence T, Zhang L. IoTNet: An efficient and accurate convolutional neural network for IoT devices. Sensors. 2019;19(24):5541.

[110] Kodali S, Hansen P, Mulholland N, Whatmough P, Brooks D, Wei GY. Applications of deep neural networks for ultra low power IoT. In: 2017 IEEE International Conference on Computer Design (ICCD). IEEE; 2017. p. 589-92.

[111] Osman M, He J, Mokbal FMM, Zhu N, Qureshi S. Ml-lgbm: A machine learning model based on light gradient boosting machine for the detection of version number attacks in rpl-based networks. IEEE Access. 2021;9:83654-65.

[112] Pamukov ME, Poulkov VK, Shterev VA. Negative selection and neural network based algorithm for intrusion detection in IoT. In: 2018 41st International Conference on Telecommunications and Signal Processing (TSP). IEEE; 2018. p. 1-5.

[113] Dhanabal L, Shantharajah S. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. International journal of advanced research in computer and communication engineering. 2015;4(6):446-52.

[114] Mishra D, Naik B, Nayak J, Souri A, Dash PB, Vimal S. Light gradient boosting machine with optimized hyperparameters for identification of malicious access in IoT network. Digital Communications and Networks. 2022.

[115] Aubet F. IoT Traffic Traces Gathered in the DS2OS Environment; 2018. Available from: https://www.kaggle.com/datasets/francoisxa/ds2ostraffictraces/discussion.

[116] Chauhan P, Atulkar M. Selection of Tree Based Ensemble Classifier for Detecting Network Attacks in IoT. In: 2021 International Conference on Emerging Smart Computing and Informatics (ESCI). IEEE; 2021. p. 770-5.

[117] Vargaftik S, Keslassy I, Orda A, Ben-Itzhak Y. Rade: Resource-efficient supervised anomaly detection using decision tree-based ensemble methods. Machine Learning. 2021;110(10):2835-66.

[118] Jing D, Chen HB. SVM based network intrusion detection for the UNSW-NB15 dataset. In: 2019 IEEE 13th international conference on ASIC (ASICON). IEEE; 2019. p. 1-4.

[119] Abbasi F, Naderan M, Alavi SE. Anomaly detection in Internet of Things using feature selection and classification based on Logistic Regression and Artificial Neural Network on N-BaIoT dataset. In: 2021 5th International Conference on Internet of Things and Applications (IoT). IEEE; 2021. p. 1-7.

[120] Zhao R, Gui G, Xue Z, Yin J, Ohtsuki T, Adebisi B, et al. A novel intrusion detection method based on lightweight neural network for internet of things. IEEE Internet of Things Journal. 2021.

[121] Lei M, Li X, Cai B, Li Y, Liu L, Kong W. P-DNN: an effective intrusion detection method based on pruning deep neural network. In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE; 2020. p. 1-9.

[122] Latif S, Zou Z, Idrees Z, Ahmad J. A novel attack detection scheme for the industrial internet of things using a lightweight random neural network. IEEE Access. 2020;8:89337-50.

[123] Rani S, Singh A, Elkamchouchi DH, Noya ID. Lightweight Hybrid Deep Learning Architecture and Model for Security in IIOT. Applied Sciences. 2022;12(13):6442.

[124] Goodfellow IJ, Shlens J, Szegedy C. Explaining and Harnessing Adversarial Examples; 2015.

[125] Kurakin A, Goodfellow I, Bengio S. Adversarial Machine Learning at Scale; 2017.

[126] Hosseini H, Xiao B, Jaiswal M, Poovendran R. On the limitation of convolutional neural networks in recognizing negative images. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE; 2017. p. 352-8.

[127] Athalye A, Carlini N, Wagner D. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In: Dy J, Krause A, editors. Proceedings of the 35th International Conference on Machine Learning. vol. 80 of Proceedings of Machine Learning Research. PMLR; 2018. p. 274-83. Available from: https://proceedings.mlr.press/v80/athalye18a.html.

[128] Shafahi A, Huang WR, Najibi M, Suciu O, Studer C, Dumitras T, et al. Poison frogs! targeted clean-label poisoning attacks on neural networks. Advances in neural information processing systems. 2018;31.

[129] Pitropakis N, Panaousis E, Giannetsos T, Anastasiadis E, Loukas G. A taxonomy and survey of attacks against machine learning. Computer Science Review. 2019;34:100199.

[130] Zhang C, Bengio S, Hardt M, Recht B, Vinyals O. Understanding deep learning (still) requires rethinking generalization. Communications of the ACM. 2021;64(3):107-15.

[131] Kurakin A, Goodfellow I, Bengio S. Adversarial machine learning at scale. arXiv preprint arXiv:161101236. 2016.

[132] Athalye A, Carlini N, Wagner D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In: International conference on machine learning. PMLR; 2018. p. 274-83.

[133] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). IEEE; 2016. p. 372-87.

[134] Aloraini F, Javed A, Rana O, Burnap P. Adversarial machine learning in IoT from an insider point of view. Journal of Information Security and Applications. 2022;70:103341.

[135] Ibitoye O, Shafiq O, Matrawy A. Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks. In: 2019 IEEE global communications conference (GLOBECOM). IEEE; 2019. p. 1-6.

[136] Pujari M, Pacheco Y, Cherukuri B, Sun W. A Comparative Study on the Impact of Adversarial Machine Learning Attacks on Contemporary Intrusion Detection Datasets. SN Computer Science. 2022;3(5):1-12.

[137] Abou Khamis R, Matrawy A. Evaluation of adversarial training on different types of neural networks in deep learning-based idss. In: 2020 international symposium on networks, computers and communications (ISNCC). IEEE; 2020. p. 1-6.

[138] Tramèr F, Kurakin A, Papernot N, Goodfellow I, Boneh D, McDaniel P. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:170507204. 2017.

[139] Yang Q, Liu Y, Cheng Y, Kang Y, Chen T, Yu H. Federated learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2019;13(3):1-207.

[140] Preuveneers D, Rimmer V, Tsingenopoulos I, Spooren J, Joosen W, Ilie-Zudor E. Chained anomaly detection models for federated learning: An intrusion detection case study. Applied Sciences. 2018;8(12):2663.

[141] Lim WYB, Luong NC, Hoang DT, Jiao Y, Liang YC, Yang Q, et al. Federated learning in mobile edge networks: A comprehensive survey. IEEE Communications Surveys & Tutorials. 2020;22(3):2031-63.

[142] Imteaj A, Thakker U, Wang S, Li J, Amini MH. A Survey on Federated Learning for Resource-Constrained IoT Devices. IEEE Internet of Things Journal. 2021.

[143] Nguyen TD, Marchal S, Miettinen M, Fereidooni H, Asokan N, Sadeghi AR. DÏoT: A federated self-learning anomaly detection system for IoT. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE; 2019. p. 756-67.

[144] Liu Y, Kumar N, Xiong Z, Lim WYB, Kang J, Niyato D. Communication-efficient federated learning for anomaly detection in industrial internet of things. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE; 2020. p. 1-6.

[145] Jiang Y, Wang S, Valls V, Ko BJ, Lee WH, Leung KK, et al. Model pruning enables efficient federated learning on edge devices. arXiv preprint arXiv:190912326. 2019.

[146] Bonawitz K, Eichner H, Grieskamp W, Huba D, Ingerman A, Ivanov V, et al. Towards federated learning at scale: System design. arXiv preprint arXiv:190201046. 2019.

[147] Popoola SI, Ande R, Adebisi B, Gui G, Hammoudeh M, Jogunola O. Federated deep learning for zero-day botnet attack detection in IoT edge devices. IEEE Internet of Things Journal. 2021.

[148] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. the Journal of machine Learning research. 2011;12:2825-30.

[149] Pedregosa F. Memory-profiler: a module for monitoring memory usage of a Python program; 2019. Available from: https://github.com/pythonprofilers/memory_profiler.

[150] Zakariyya I. Reducing computational cost of running AIS for IoT cybersecurity.; 2019. Available from: https://github.com/izakariyya/ais.

[151] Bisong E. Introduction to Scikit-learn. In: Building machine learning and deep learning models on Google cloud platform. Springer; 2019. p. 215-29.

[152] Zebari R, Abdulazeez A, Zeebaree D, Zebari D, Saeed J. A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. Journal of Applied Science and Technology Trends. 2020;1(2):56-70.

[153] Lam LHT, Chu NT, Tran TO, Do DT, Le NQK. A radiomics-based machine learning model for prediction of tumor mutational burden in lower-grade gliomas. Cancers. 2022;14(14):3492.

[154] Natekin A, Knoll A. Gradient boosting machines, a tutorial. Frontiers in neurorobotics. 2013;7:21.

[155] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825-30.

[156] Raybaut P. Spyder: Scientific python development environment, 2009–. URL" https://github com/spyder-ide/spyder"[Online. 2017.

[157] Zakariyya I. Resource Efficient IoT LGBM Algorithm.; 2020. Available from: https://github.com/izakariyya/Resource_Constraint_Algorithm.

[158] Sarker IH, Kayes A, Badsha S, Alqahtani H, Watters P, Ng A. Cybersecurity data science: an overview from machine learning perspective. Journal of Big data. 2020;7(1):1-29.

[159] Larriva-Novo XA, Vega-Barbas M, Villagrá VA, Rodrigo MS. Evaluation of cybersecurity data set characteristics for their applicability to neural networks algorithms detecting cybersecurity anomalies. IEEE Access. 2020;8:9005-14.

[160] Abiodun OI, Jantan A, Omolara AE, Dada KV, Mohamed NA, Arshad H. State-of-the-art in artificial neural network applications: A survey. Heliyon. 2018;4(11):e00938.

[161] Oyama Y, Ben-Nun T, Hoefler T, Matsuoka S. Accelerating deep learning frameworks with micro-batches. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE; 2018. p. 402-12.

[162] Huang Y, Cheng Y, Bapna A, Firat O, Chen D, Chen M, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in neural information processing systems. 2019;32:103-12.

[163] Han S, Pool J, Tran J, Dally WJ. Learning both Weights and Connections for Efficient Neural Networks; 2015.

[164] Johansson R. Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib. Apress; 2018.

[165] Komer B, Bergstra J, Eliasmith C. Hyperopt-sklearn. In: Automated Machine Learning. Springer, Cham; 2019. p. 97-111.

[166] Bosman A, Engelbrecht A, Helbig M. Fitness landscape analysis of weight-elimination neural networks. Neural Processing Letters. 2018;48(1):353-73.

[167] Ide H, Kurita T. Improvement of learning for CNN with ReLU activation by sparse regularization. In: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE; 2017. p. 2684-91.

[168] Pumperla M. Hyperas: Hyperopt: A very simple and convenient wrapper for hyperparameter optimization; 2018. Available from: https://github.com/maxpumperla/hyperas.

[169] Zakariyya I. Resource Efficient IoT DNNs Algorithm.; 2021. Available from: https://github.com/izakariyya/R_DNN_IoT.

[170] Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, et al. Mixed Precision Training. In: International Conference on Learning Representations; 2018. .

[171] TensorFlow. float16 Quantization.. TensorFlow; 2022. Available from: https://www.tensorflow.org/lite/performance/post_training_float16_quant.

[172] Zhao J, Dai S, Venkatesan R, Liu M, Khailany B, Dally B, et al. Low-Precision Training in Logarithmic Number System using Multiplicative Weight Update. CoRR. 2021;abs/2106.13914. Available from: https://arxiv.org/abs/2106.13914.

[173] intel. Choose Precision.. Intel Corporation; 2020. Available from: https://software.intel.com/content/www/us/en/develop/articles/should-i-choose-fp16-or-fp32-for-my-deep-learning-model.html.

[174] Catak FO, Balaban ME. CloudSVM: training an SVM classifier in cloud computing systems. In: Joint International Conference on Pervasive Computing and the Networked World. Springer; 2012. p. 57-68.

[175] Jose C, Goyal P, Aggrwal P, Varma M. Local deep kernel learning for efficient non-linear svm prediction. In: International conference on machine learning. PMLR; 2013. p. 486-94.

[176] Papernot N, Faghri F, Carlini N, Goodfellow I, Feinman R, Kurakin A, et al.. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library; 2018.

[177] Suciu O, Coull SE, Johns J. Exploring adversarial examples in malware detection. In: 2019 IEEE Security and Privacy Workshops (SPW). IEEE; 2019. p. 8-14.

[178] Dunn C, Moustafa N, Turnbull B. Robustness evaluations of sustainable machine learning models against data poisoning attacks in the internet of things. Sustainability. 2020;12(16):6434.

[179] Lever J, Krzywinski M, Altman N. Points of significance: Regularization. Nature methods. 2016;13(10):803-5.

[180] Alazab M, RM SP, Parimala M, Maddikunta PKR, Gadekallu TR, Pham QV. Federated Learning for Cybersecurity: Concepts, Challenges, and Future Directions. IEEE Transactions on Industrial Informatics. 2021;18(5):3501-9.

[181] Yin B, Yin H, Wu Y, Jiang Z. FDC: A secure federated deep learning mechanism for data collaborations in the Internet of Things. IEEE Internet of Things Journal. 2020;7(7):6348-59.

[182] He C, Mushtaq E, Ding J, Avestimehr S. FedNAS: Federated Deep Learning via Neural Architecture Search; 2022. Available from: https://openreview.net/forum?id=1OHZX4YDqhT.

[183] Baldominos A, Saez Y, Isasi P. A survey of handwritten character recognition with mnist and emnist. Applied Sciences. 2019;9(15):3169.

[184] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems. 2019;32:8026-37.

[185] Ryffel T, Trask A, Dahl M, Wagner B, Mancuso J, Rueckert D, et al. A generic framework for privacy preserving deep learning. arXiv preprint arXiv:181104017. 2018.

[186] Zakariyya. Resource Efficient Federated Algorithm with Virtual Workers.; 2022. Available from: https://github.com/izakariyya/sim-virtual-fed-dnn.

[187] Zakariyya I. Resource Efficient Federated Algorithm with realistic Workers.; 2022. Available from: https://github.com/izakariyya/testbd-fl-iot.

[188] Krueger D, Memisevic R. Regularizing rnns by stabilizing activations. arXiv preprint arXiv:151108400. 2015.

[189] Sun X, Wang N, Chen CY, Ni J, Agrawal A, Cui X, et al. Ultra-low precision 4-bit training of deep neural networks. Advances in Neural Information Processing Systems. 2020;33.

Appendix A

Dissemination

1. Idris Zakariyya; Al-Kadri, M. Omar; Kalutarage, Harsha; Petrovski, Andrei, Reducing Computational Cost in IoT Cyber Security: Case Study of Artificial Immune System Algorithm. SECRYPT, 2019: 523-528. https://dblp.org/rec/conf/icete/ZakariyyaAKP19 (chapter 3)

2. Idris Zakariyya; Al-Kadri, M. Omar; Kalutarage, Harsha, Resource Efficient Boosting Method for IoT Security Monitoring, 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), 2021, pp. 1-6, doi: 10.1109/CCNC49032.2021.9369620. (chapter 3)

3. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Robust, Effective and Resource Efficient Deep Neural Network for Intrusion Detection in IoT Networks, In Proceedings of the 8th ACM Cyber-Physical System Security Workshop (CPSS'22), 2022, https://doi.org/10.1145/3494107.3522772 (chapter 4 and 5)

4. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Memory Efficient Federated Deep Learning for Intrusion Detection in IoT Networks, AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, http://ceur-ws.org/Vol-3125/paper7.pdf (chapter 6)

5. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Resource Efficient Federated Deep Learning for IoT Security Monitoring, Book Chapter in ADIoT, in 27th European Symposium on Research in Computer Security (ESORICS 2022), pp 122–142, Lecture Notes in Computer Science, vol 13745. Springer, Cham.

https://link.springer.com/chapter/10.1007/978-3-031-21311-3_6. (chapter 6)

6. Idris Zakariyya; Harsha, Kalutarage; M. Omar, Al-Kadri, Towards a Robust, Effective and Resource Efficient Deep Learning Technique for IoT Security Monitoring, Computers & Security, Under Review.